
Miranda Documentation

Release 0.5.0

Trevor James Smith

Jun 19, 2023

CONTENTS

1	Features	3
2	Installation	5
3	Contributing	7
3.1	Contents:	7
3.2	Feedback	73
	Python Module Index	75
	Index	77

Python utilities for climate data collection, conversion, and management

- Documentation:
- Free Software:

FEATURES

Data collection functions for climate and forecast data hosted at:

- ECMWF (ERA5, ERA5-Land, TIGGE)
- ECCC (Canada) (Monthly Climate Summaries, ECCC GEOAPI - In development)
- NCAR (CORDEX-NA on AWS)

Data conversion for [Climate and Forecasting \(CF\)](#) Variable and Metadata compliance:

- ECMWF (ERA5, ERA5-Land, TIGGE - In Development)
- ECCC (Canada) (Flat File Observations, Monthly Climate Summaries, Adjusted and Homogenized Climate Data, ECCC GEOAPI - In Development)
- MELCC (Québec)
- Hydro-Québec (In Development)

Database structuring and facets validation:

- **Simulations:**
 - WCRP (CMIP5, CMIP6, CMIP5-CORDEX, CORDEX-ADJUST, ISIMIP, etc.)
- **Station-Observations:**
 - MELCC (Québec) (Needs [mdbtools](#) installed)
 - ECCC (Canada) (In Development)
 - Hydro-Québec (In Development)
- **Gridded-Observations:**
 - NRCAN (Canada) (Future)
 - MELCC (Future)
- **Reanalyses:**
 - ECMWF (ERA5, ERA5-Land, TIGGE)
 - NASA (DayMET, AgMerra/AgCFSR, MERRA2) - In Development
 - NCEP (CFSR/CFSv2) - In Development
 - WFDEI-GEM-CaPa (University of Saskatchewan) - In Development

INSTALLATION

miranda can be installed from PyPI:

```
$ pip install miranda
```

Some functionalities require complex-to-install dependencies. In order to gain access to them, we strongly suggest using [Anaconda](#) to manage your environment:

```
$ conda env create -f environment.yml
$ conda activate miranda
$ pip install miranda[full]
```

For more information about Anaconda/Miniconda/conda-forge:

- Miniconda: <https://docs.conda.io/en/latest/miniconda.html>
- conda-forge: <https://conda-forge.org/#about>

miranda also relies on [PyESSV](#) for its climate data controlled vocabulary. This library is optional for users who do not require validation checks, but enabling this feature requires additional installation steps:

```
$ mkdir -p ~/.esdoc
$ git clone git@github.com:ES-DOC/pyessv-archive.git ~/.esdoc/pyessv-archive
```


CONTRIBUTING

See the contributing documentation: <https://miranda.readthedocs.io/en/latest/contributing.html>

3.1 Contents:

3.1.1 Installation

At the command line, enter:

```
$ pip install miranda
```

This is the preferred method to install miranda, as it will always install the most recent stable release.

To make use of remote operations (*miranda.remote*) and some dataset downloading functions (*miranda.ncar miranda.ecmwf*), additional libraries are needed. They can be installed with the following:

```
$ pip install miranda[remote]
```

For better RAM usage when converting datasets, some additional/optional GIS libraries can be installed as well:

```
$ pip install miranda[gis]
```

If you don't have `pip` installed, this [Python installation guide](#) can guide you through the process.

From sources

The sources for miranda can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/Ouranosinc/miranda
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/Ouranosinc/miranda/tarball/main
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Alternatively, you can also install a local copy via `pip`:

```
$ pip install .
```

Creating a Conda environment

To create a conda development environment including all miranda dependencies, enter the following command from within your cloned repo:

```
$ conda create -n my_miranda_env python=3.8 --file=environment.yml
$ conda activate my_miranda_env
$ pip install -e .[dev]
```

3.1.2 Usage

To use Miranda in a project:

```
import miranda
```

3.1.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at <https://github.com/Ouranosinc/miranda/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

Miranda could always use more documentation, whether as part of the official Miranda docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/Ouranosinc/miranda/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here’s how to set up *miranda* for local development.

1. [Fork](#) the *miranda* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/miranda.git
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. Begin by installing a development build of your branch:

```
# To install miranda with its development environment dependencies
$ pip install -e .[dev]
# To install miranda with GIS libraries
$ pip install -e .[gis]
# To install miranda with its documentation dependencies
$ pip install -e .[docs]
# To install miranda with its remote API dependencies
$ pip install -e .[remote]
```

5. When you’re done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

6. Commit your changes and push your branch to GitHub:

```
$ git add .  
$ git commit -m "Your detailed description of your changes."  
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7, 3.8, 3.9, 3.10, and 3.11. Check <https://github.com/Ouranosinc/miranda/actions> for active pull request builds or run the `tox` command and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ pytest test/test_miranda.py
```

3.1.4 Credits

Development Lead

- Trevor James Smith <smith.trevorj@ouranos.ca> @Zeitsperre

Co-Developers

- Pascal Bourgault <bourgault.pascal@ouranos.ca> @aulemahal
- Travis Logan <logan.travis@ouranos.ca> @tlogan2000

Contributors

- Sébastien Biner <biner.sebastien@hydroquebec.com> @sbiner
- David Huard <huard.david@ouranos.ca> @huard
- Gabriel Rondeau-Genesse <rondeau-genesse.gabriel@ouranos.ca> @RondeauG

3.1.5 History

v0.5.0 (2023-06-19)

Contributors to this version: Juliette Lavoie (:user: [juliettelavoie](#)), Trevor James Smith (@Zeitsperre).

New features

- Added support for collecting and converting *ptype* ECMWF ERA5 variable.
- A new "`_frequency`": `true` toggle for returning the output frequency of converted data.
- Added a new JSON template for NEX-GDDP-CMIP6 datasets.
- *miranda* is now [PEP 517](#) and [PEP 621](#) compliant, using the [flit](#) backend.

Internal changes

- Various fixes to existing docstrings.
- Time frequency checks are more resilient when converting Monthly time-step data.
- Masking and regridding of datasets when running `convert_dataset` is now optional or automatic.
- Updated templates to newest API.
- Created a *gis* recipe for exclusively installing GIS libraries.
- Removed many unneeded dependencies, cleaned up Makefile.
- All public-facing functions now contain at least a minimal docstring for documentation generation.

v0.4.0 (2023-03-30)

Contributors to this version: Trevor James Smith (@Zeitsperre), Pascal Bourgault (@aulemahal), Travis Logan (@tlogan2000).

New features

- Improvements have been made to the development documentation; Project URLs, ReadTheDocs theming, and other quality of life changes.
- Conversion JSON definitions now support pre-processing to render dimensions and variable names consistent before running corrections/conversions.
- **New datasets with CF-like attributes conversion supported:**
 - RDRS (ECCC)
 - GRNCH (ETS)
- Preliminary `miranda.io` module for organizing output-writing functionality.
- New `miranda.io.fetch_chunk_config` function for “rechunking” datasets according to project presets.
- New `mirands.io.utils.name_output_file` for generating names from Dataset facets or from a dictionary.
- New `mirands.gis.subset_domain` for clipping dataset to a preconfigured region.

Bug fixes

- Many data-related utilities now have more accurate static typing.
- Converted dataset global attributes are now synchronized for consistency.
- ECMWF-based datasets now implement more consistent conversion factors and metadata.
- `miranda.storage.file_size` now handles dictionaries of Pathlib objects.

Internal changes

- Pre-commit version updates.
- Improvements have been made to the development documentation; Project URLs, ReadTheDocs theming, installation methods, and other quality of life changes.
- **Schema and folder structure updates:**
 - *gridded-obs -> reconstruction*
 - *bias-adjust-project* is used when present and not just when *level=="biasadjusted"*
- CI now using *tox>=4.0* and *ubuntu-latest* virtual machine images.

v0.3.0 (2022-11-24)

Contributors to this version: Trevor James Smith ([@Zeitsperre](#)), Pascal Bourgault ([@aulemahal](#)), David Huard ([@huard](#)), Travis Logan ([@tlogan2000](#)), Gabriel Rondeau-Genesse ([@RondeauG](#)), and Sébastien Biner ([@sbiner](#)).

Announcements

- First public release on PyPI.

New features

- **Dataset conversion tools (`miranda.convert`) use a JSON-definition file to dynamically populate metadata, run data quality checks, and convert units to CF-compliant standard. Supported datasets are:**
 - ERA5/ERA5-Land (complete)
 - MELCC (stations) (beta)
 - ECCC (stations) (alpha)
 - NASA DayMet (WIP)
 - NASA AgMerra/AgCFSR (WIP)
 - Hydro Québec (stations) (WIP)
 - DEH (stations) (WIP)
 - WFDEI-GEM-CAPA (WIP)
- Module (`miranda.eccc`) for ECCC station data and ECCC Adjusted and Homogenized Canadian Climate Data (AHCCD) conversion (WIP).
- Module (`miranda.ncar`) for fetching interpolated CORDEX-NAM (22i/44i) from NCAR AWS data storage.

- Module (`miranda.ecmwf`) for fetching ECMWF ERA5/-Land (single-levels, pressure-levels, monthly-means) datasets via CDSAPI.
- Module (`miranda.gis`) for setting specific subsetting domains used when converting gridded datasets.
- Modules (`miranda.archive` and `miranda.remote`) for performing data archiving actions locally and remotely (powered by `fabric` and `paramiko`) (WIP).
- **Module (`miranda.decode`) for ingesting and parsing dataset metadata based on filename and dataset attributes. Supported datasets are:**
 - *miranda* converted datasets
 - CMIP6
 - CMIP5
 - CMIP5-CORDEX
 - ISIMIP-FT
 - CanDCS-U6 (PCIC)
- Module (`miranda.structure`) for create constructing file-tree databases based on YAML-defined metadata schemas (WIP).
- Modules (`miranda.cv` and `miranda.validators`) for validating metadata using ESGF controlled vocabularies (taken from `pyessv-archive`) and schema definitions (powered by `schema`), respectively (WIP).

3.1.6 License

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation

(continues on next page)

(continued from previous page)

source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their

(continues on next page)

(continued from previous page)

Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work

(continues on next page)

(continued from previous page)

by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate

(continues on next page)

(continued from previous page)

```
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.
```

Copyright 2019 Trevor James Smith, Ouranos Inc., and contributors

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

3.1.7 miranda

miranda package

Copyright 2019-2023 Trevor James Smith and Ouranos Inc.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<https://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

```
class miranda.Database(source: *, destination: Path | str | None = None, common_path: Path | str | None =
    None, file_pattern: str | list[str] = '*.nc', project_name: str | None = None, recursive:
    bool = True)
```

Bases: object

Database management class.

archive()

Not yet implemented.

```
group_by(*, common_path: Path | str | None = None, subdirectories: bool = True, dates: bool = True, size:
    int = 10737418240)
```

Grouping meta-function.

Notes

Not yet implemented.

items()

Show items.

keys()

Show keys.

target(*target*: [Path](#) | *str*)

Target directory or server address.

transfer()

Not yet implemented.

values()

Show values.

class `miranda.FileMeta`(*path*: *str*, *size*: *int* = -1)

Bases: `object`

File path and size.

```
django = {'path': ['CharField', 'max_length=512'], 'size': ['IntegerField',
'null=True', 'blank=True']}
```

class `miranda.StorageState`(*base_path*, *capacity*=-1, *used_space*=-1, *free_space*=-1)

Bases: `object`

Information regarding the storage capacity of a disk.

Subpackages

`miranda.archive` package

Archive module.

`miranda.archive.group_by_deciphered_date`(*files*: *generator* | *list*[*str* | [pathlib.Path](#)]) → *dict*[*str*, *list*[[pathlib.Path](#)]]

Find a common date and groups files based on year and month.

Parameters

files (*GeneratorType* or *list* of *str* or [pathlib.Path](#))

Returns

dict[*str*, *list*[[pathlib.Path](#)]]

`miranda.archive.group_by_length`(*files*: *generator* | *list*[*str* | [pathlib.Path](#)], *size*: *int* = 10, *sort*: *bool* = *False*) → *list*[*list*[[pathlib.Path](#)]]

Group files by an arbitrary number of file entries.

Parameters

- **files** (*GeneratorType* or *list* of *str* or [pathlib.Path](#))
- **size** (*int*)
- **sort** (*bool*)

Returns*list[list[pathlib.Path]]*

`miranda.archive.group_by_size(files: generator | list[str | pathlib.Path], size: int = 10737418240) → list[list[pathlib.Path]]`

Group files up until a desired size and save it as a grouping within a list.

Parameters

- **files** (*GeneratorType or list of str or [pathlib.Path](#)*)
- **size** (*int*)

Returns*list[list[pathlib.Path]]*

`miranda.archive.group_by_subdirectories(files: generator | list[str | pathlib.Path], within: Path | str | None = None) → dict[str, list[pathlib.Path]]`

Group files based on the parent folder that they are located within.

Parameters

- **files** (*GeneratorType or list of str or [pathlib.Path](#)*)
- **within** (*str or [pathlib.Path](#)*)

Returns*dict[str, list[pathlib.Path]]*

`miranda.archive.select_by_date_modified(source: str | Path, year: int | None, month: int | None, day: int | None, *, suffixes: str = 'nc', date: datetime) → list[pathlib.Path]`

Select files by the date on which they were last modified.

Parameters

- **source** (*str or Path*)
- **year** (*int*)
- **month** (*int*)
- **day** (*int*)
- **suffixes** (*str*)
- **date** (*datetime.date*)

Returns*list of Path***Submodules****[miranda.archive._groupings module](#)**

`miranda.archive._groupings.group_by_deciphered_date(files: generator | list[str | pathlib.Path]) → dict[str, list[pathlib.Path]]`

Find a common date and groups files based on year and month.

Parameters

files (*GeneratorType or list of str or [pathlib.Path](#)*)

Returns

dict[str, list[pathlib.Path]]

`miranda.archive._groupings.group_by_length(files: generator | list[str | pathlib.Path], size: int = 10, sort: bool = False) → list[list[pathlib.Path]]`

Group files by an arbitrary number of file entries.

Parameters

- **files** (*GeneratorType or list of str or [pathlib.Path](#)*)
- **size** (*int*)
- **sort** (*bool*)

Returns

list[list[[pathlib.Path](#)]]

`miranda.archive._groupings.group_by_size(files: generator | list[str | pathlib.Path], size: int = 10737418240) → list[list[pathlib.Path]]`

Group files up until a desired size and save it as a grouping within a list.

Parameters

- **files** (*GeneratorType or list of str or [pathlib.Path](#)*)
- **size** (*int*)

Returns

list[list[[pathlib.Path](#)]]

`miranda.archive._groupings.group_by_subdirectories(files: generator | list[str | pathlib.Path], within: Path | str | None = None) → dict[str, list[pathlib.Path]]`

Group files based on the parent folder that they are located within.

Parameters

- **files** (*GeneratorType or list of str or [pathlib.Path](#)*)
- **within** (*str or [pathlib.Path](#)*)

Returns

dict[str, list[[pathlib.Path](#)]]

miranda.archive._selection module

`miranda.archive._selection.select_by_date_modified(source: str | Path, year: int | None, month: int | None, day: int | None, *, suffixes: str = 'nc', date: datetime) → list[pathlib.Path]`

Select files by the date on which they were last modified.

Parameters

- **source** (*str or Path*)
- **year** (*int*)
- **month** (*int*)
- **day** (*int*)
- **suffixes** (*str*)

- **date** (*datetime.date*)

Returns

list of Path

miranda.convert package

Data Conversion module.

`miranda.convert.aggregate(ds: Dataset, freq: str = 'day') → dict[str, xarray.core.dataset.Dataset]`

Parameters

- **ds** (*xarray.Dataset*)
- **freq** (*str*)

Returns

dict[str, xarray.Dataset]

`miranda.convert.aggregations_possible(ds: Dataset, freq: str = 'day') → dict[str, set[str]]`

Determine which aggregations are possible based on variables within a dataset.

Parameters

- **ds** (*xarray.Dataset*)
- **freq** (*str*)

Returns

dict[str, set[str]]

`miranda.convert.dataset_conversion(input_files: str | PathLike | Sequence[str | PathLike] | Iterator[PathLike] | Dataset, project: str, domain: str | None = None, mask: Dataset | DataArray | None = None, mask_cutoff: float | bool = False, regrid: bool = False, add_version_hashes: bool = True, preprocess: Callable | str | None = 'auto', **xr_kwargs) → Dataset | DataArray`

Convert an existing Xarray-compatible dataset to another format with variable corrections applied.

Parameters

- **input_files** (*str or os.PathLike or Sequence[str or os.PathLike] or Iterator[os.PathLike] or xr.Dataset*) – Files or objects to be converted. If sent a list or GeneratorType, will open with `xarray.open_mfdataset()` and concatenate files.
- **project** (*{“cordex”, “cmip5”, “cmip6”, “ets-grnc”, “isimip-ft”, “pcic-candcs-u6”, “converted”}*) – Project name for decoding/handling purposes.
- **domain** (*{“global”, “nam”, “can”, “qc”, “mtl”}, optional*) – Domain to perform subsetting for. Default: None.
- **mask** (*Optional[Union[xr.Dataset, xr.DataArray]]*) – DataArray or single data_variable dataset containing mask.
- **mask_cutoff** (*float or bool*) – If `land_sea_mask` supplied, the threshold above which to mask with `land_sea_mask`. Default: False.
- **regrid** (*bool*) – Performing regridding with `xesmf`. Default: False.
- **add_version_hashes** (*bool*) – If True, version name and sha256sum of source file(s) will be added as a field among the global attributes.

- **preprocess** (*callable or str, optional*) – Preprocessing functions to perform over each Dataset. Default: “auto” - Run preprocessing fixes based on supplied fields from metadata definition. Callable - Runs function over Dataset (single) or supplied to *preprocess* (multifile dataset).
- ****xr_kwargs** – Arguments passed directly to xarray.

Returns

xr.Dataset or xr.DataArray

`miranda.convert.dataset_corrections(ds: Dataset, project: str) → Dataset`

Convert variables to CF-compliant format

`miranda.convert.dims_conversion(d: Dataset, p: str, m: dict) → Dataset`

Rename dimensions to CF to their equivalents.

Parameters

- **d** (*xarray.Dataset*) – Dataset with dimensions to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns

xarray.Dataset

`miranda.convert.gather_agcfsr(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather agCFSR source data.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[[pathlib.Path](#)]]

`miranda.convert.gather_agmerra(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather agMERRA source data.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[[pathlib.Path](#)]]

`miranda.convert.gather_ecmwf(project: str, path: str | PathLike, back_extension: bool = False, monthly_means: bool = False) → dict[str, list[pathlib.Path]]`

Parameters

- **project** (*{“era5-single-levels”, “era5-pressure-levels”, “era5-land”}*)
- **path** (*str or os.PathLike*)
- **back_extension** (*bool*)
- **monthly_means** (*bool*)

Returns

dict[str, list[[pathlib.Path](#)]]

`miranda.convert.gather_emdna(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather raw EMDNA files for preprocessing.

Put all files with the same member together.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert.gather_grnch(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather raw ETS-GRNCH files for preprocessing.

Parameters

path (*str or os.PathLike*)

Returns

dict(str, dict(str, list[Path])) or None

`miranda.convert.gather_nex(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather raw NEX files for preprocessing.

Put all files that should be contained in one dataset in one entry of the dictionary.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert.gather_nrcan_gridded_obs(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather NRCan Gridded Observations source data.

Parameters

path (*str or os.PathLike*)

Returns

dict(str, list[pathlib.Path])

`miranda.convert.gather_raw_rdrs_by_years(path: str | PathLike) → dict[str, dict[str, list[pathlib.Path]]]`

Gather raw RDRS files for preprocessing.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, dict[str, list[pathlib.Path]]]

`miranda.convert.gather_rdrs(name: str, path: str | PathLike, suffix: str, key: str) → dict[str, dict[str, list[pathlib.Path]]]`

Gather RDRS processed source data.

Parameters

- **name** (*str*)
- **path** (*str or os.PathLike*)
- **suffix** (*str*)
- **key** (*{“raw”, “cf”}*) – Indicating which variable name dictionary to search for.

Returns

dict[str, list[pathlib.Path]]

`miranda.convert.gather_sc_earth(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather SC-Earth source data

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert.gather_wfdei_gem_capa(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather WFDEI-GEM-CaPa source data.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert.load_json_data_mappings(project: str) → dict[str, Any]`

Load JSON mappings for supported dataset conversions.

Parameters

project (*str*)

Returns

dict[str, Any]

`miranda.convert.metadata_conversion(d: Dataset, p: str, m: dict) → Dataset`

Update xarray dataset and data_vars with project-specific metadata fields.

Parameters

- **d** (*xarray.Dataset*) – Dataset with metadata to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns

xarray.Dataset

`miranda.convert.threshold_mask(ds: Dataset | DataArray, *, mask: Dataset | DataArray, mask_cutoff: float | bool = False) → Dataset | DataArray`

Land-Sea mask operations.

Parameters

- **ds** (*xr.Dataset or str or os.PathLike*)
- **mask** (*xr.Dataset or xr.DataArray*)
- **mask_cutoff** (*float or bool*)

Returns

xr.Dataset or xr.DataArray

`miranda.convert.variable_conversion(d: Dataset, p: str, m: dict) → Dataset`

Add variable metadata and remove nonstandard entries.

Parameters

- **d** (*xarray.Dataset*) – Dataset with variable(s) to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns*xarray.Dataset***Submodules****miranda.convert._aggregation module**

Aggregation module.

`miranda.convert._aggregation.aggregate(ds: Dataset, freq: str = 'day') → dict[str, xarray.core.dataset.Dataset]`

Parameters

- **ds** (*xarray.Dataset*)
- **freq** (*str*)

Returns*dict[str, xarray.Dataset]*

`miranda.convert._aggregation.aggregations_possible(ds: Dataset, freq: str = 'day') → dict[str, set[str]]`

Determine which aggregations are possible based on variables within a dataset.

Parameters

- **ds** (*xarray.Dataset*)
- **freq** (*str*)

Returns*dict[str, set[str]]***miranda.convert._data_corrections module**

`miranda.convert._data_corrections.dataset_conversion(input_files: str | PathLike | Sequence[str | PathLike] | Iterator[PathLike] | Dataset, project: str, domain: str | None = None, mask: Dataset | DataArray | None = None, mask_cutoff: float | bool = False, regrid: bool = False, add_version_hashes: bool = True, preprocess: Callable | str | None = 'auto', **xr_kwargs) → Dataset | DataArray`

Convert an existing Xarray-compatible dataset to another format with variable corrections applied.

Parameters

- **input_files** (*str or os.PathLike or Sequence[str or os.PathLike] or Iterator[os.PathLike] or xr.Dataset*) – Files or objects to be converted. If sent a list or GeneratorType, will open with `xarray.open_mfdataset()` and concatenate files.
- **project** (*{“cortex”, “cmip5”, “cmip6”, “ets-grnch”, “isimip-ft”, “pcic-candcs-u6”, “converted”}*) – Project name for decoding/handling purposes.

- **domain** (*{“global”, “nam”, “can”, “qc”, “mtl”}, optional*) – Domain to perform subsetting for. Default: None.
- **mask** (*Optional[Union[xr.Dataset, xr.DataArray]]*) – DataArray or single data_variable dataset containing mask.
- **mask_cutoff** (*float or bool*) – If land_sea_mask supplied, the threshold above which to mask with land_sea_mask. Default: False.
- **regrid** (*bool*) – Performing regridding with xesmf. Default: False.
- **add_version_hashes** (*bool*) – If True, version name and sha256sum of source file(s) will be added as a field among the global attributes.
- **preprocess** (*callable or str, optional*) – Preprocessing functions to perform over each Dataset. Default: “auto” - Run preprocessing fixes based on supplied fields from metadata definition. Callable - Runs function over Dataset (single) or supplied to *preprocess* (multifile dataset).
- ****xr_kwargs** – Arguments passed directly to xarray.

Returns

xr.Dataset or xr.DataArray

`miranda.convert._data_corrections.dataset_corrections(ds: Dataset, project: str) → Dataset`

Convert variables to CF-compliant format

`miranda.convert._data_corrections.dims_conversion(d: Dataset, p: str, m: dict) → Dataset`

Rename dimensions to CF to their equivalents.

Parameters

- **d** (*xarray.Dataset*) – Dataset with dimensions to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns

xarray.Dataset

`miranda.convert._data_corrections.load_json_data_mappings(project: str) → dict[str, Any]`

Load JSON mappings for supported dataset conversions.

Parameters

project (*str*)

Returns

dict[str, Any]

`miranda.convert._data_corrections.metadata_conversion(d: Dataset, p: str, m: dict) → Dataset`

Update xarray dataset and data_vars with project-specific metadata fields.

Parameters

- **d** (*xarray.Dataset*) – Dataset with metadata to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns

xarray.Dataset

`miranda.convert._data_corrections.threshold_mask(ds: Dataset | DataArray, *, mask: Dataset | DataArray, mask_cutoff: float | bool = False) → Dataset | DataArray`

Land-Sea mask operations.

Parameters

- **ds** (*xr.Dataset or str or os.PathLike*)
- **mask** (*xr.Dataset or xr.DataArray*)
- **mask_cutoff** (*float or bool*)

Returns

xr.Dataset or xr.DataArray

`miranda.convert._data_corrections.variable_conversion(d: Dataset, p: str, m: dict) → Dataset`

Add variable metadata and remove nonstandard entries.

Parameters

- **d** (*xarray.Dataset*) – Dataset with variable(s) to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns

xarray.Dataset

`miranda.convert._data_definitions module`

`miranda.convert._data_definitions.gather_agcfsr(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather agCFSR source data.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[[pathlib.Path](#)]]

`miranda.convert._data_definitions.gather_agmerra(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather agMERRA source data.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[[pathlib.Path](#)]]

`miranda.convert._data_definitions.gather_ecmwf(project: str, path: str | PathLike, back_extension: bool = False, monthly_means: bool = False) → dict[str, list[pathlib.Path]]`

Parameters

- **project** (*{“era5-single-levels”, “era5-pressure-levels”, “era5-land”}*)
- **path** (*str or os.PathLike*)
- **back_extension** (*bool*)
- **monthly_means** (*bool*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert._data_definitions.gather_emdna(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather raw EMDNA files for preprocessing.

Put all files with the same member together.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert._data_definitions.gather_grnch(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather raw ETS-GRNCH files for preprocessing.

Parameters

path (*str or os.PathLike*)

Returns

dict(str, dict(str, list[Path])) or None

`miranda.convert._data_definitions.gather_nex(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather raw NEX files for preprocessing.

Put all files that should be contained in one dataset in one entry of the dictionary.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert._data_definitions.gather_nrcan_gridded_obs(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather NRCan Gridded Observations source data.

Parameters

path (*str or os.PathLike*)

Returns

dict(str, list[pathlib.Path])

`miranda.convert._data_definitions.gather_raw_rdrs_by_years(path: str | PathLike) → dict[str, dict[str, list[pathlib.Path]]]`

Gather raw RDRS files for preprocessing.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, dict[str, list[pathlib.Path]]]

`miranda.convert._data_definitions.gather_rdrs(name: str, path: str | PathLike, suffix: str, key: str) → dict[str, dict[str, list[pathlib.Path]]]`

Gather RDRS processed source data.

Parameters

- **name** (*str*)
- **path** (*str or os.PathLike*)

- **suffix** (*str*)
- **key** (*{“raw”, “cf”}*) – Indicating which variable name dictionary to search for.

Returns

dict[str, list[pathlib.Path]]

`miranda.convert._data_definitions.gather_sc_earth(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather SC-Earth source data

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

`miranda.convert._data_definitions.gather_wfdei_gem_capa(path: str | PathLike) → dict[str, list[pathlib.Path]]`

Gather WFDEI-GEM-CaPa source data.

Parameters

path (*str or os.PathLike*)

Returns

dict[str, list[pathlib.Path]]

miranda.convert._reconstruction module

`miranda.convert._reconstruction.reanalysis_processing(data: dict[str, list[str | os.PathLike]],
output_folder: str | PathLike, variables:
Sequence[str], aggregate: str | bool = False,
domains: str | list[str] = '_DEFAULT', start:
str | None = None, end: str | None = None,
target_chunks: dict | None = None,
output_format: str = 'netcdf', overwrite: bool
= False, engine: str = 'h5netcdf', n_workers:
int = 4, **dask_kwargs) → None`

Parameters

- **data** (*dict[str, list[str]]*)
- **output_folder** (*str or os.PathLike*)
- **variables** (*Sequence[str]*)
- **aggregate** (*{“day”, None}*)
- **domains** (*{“QC”, “CAN”, “AMNO”, “NAM”, “GLOBAL”}*)
- **start** (*str, optional*)
- **end** (*str, optional*)
- **target_chunks** (*dict, optional*)
- **output_format** (*{“netcdf”, “zarr”}*)
- **overwrite** (*bool*)
- **engine** (*{“netcdf4”, “h5netcdf”}*)
- **n_workers** (*int*)

Returns

None

miranda.convert.deh module

DEH Hydrograph Conversion module.

`miranda.convert.deh.open_txt(path: str | Path, cf_table: dict | None = {'flag': {'comment': 'See DEH technical information for details.', 'long_name': 'data flag'}, 'q': {'long_name': 'River discharge', 'units': 'm3 s-1'}}) → Dataset`

Extract daily HQ meteorological data and convert to xr.DataArray with CF-Convention attributes.

miranda.convert.eccc module

Environment and Climate Change Canada Data Conversion module.

`miranda.convert.eccc.convert_canswe(file: str | Path, output: str | Path)`

Convert the CanSWE netCDF files to production-ready CF-compliant netCDFs.

miranda.convert.eccc_rdrs module

Environment and Climate Change Canada RDRS conversion tools.

`miranda.convert.eccc_rdrs.convert_rdrs(project: str, input_folder: str | PathLike, output_folder: str | PathLike, output_format: str = 'zarr', working_folder: str | PathLike | None = None, overwrite: bool = False, **dask_kwargs) → None`

Parameters

- **project** (*str*)
- **input_folder** (*str* or *os.PathLike*)
- **output_folder** (*str* or *os.PathLike*)
- **output_format** ({*"netcdf"*, *"zarr"*})
- **working_folder** (*str* or *os.PathLike*, *optional*)
- **overwrite** (*bool*)
- ****dask_kwargs**

Returns

None

`miranda.convert.eccc_rdrs.rdrs_to_daily(project: str, input_folder: str | PathLike, output_folder: str | PathLike, working_folder: str | PathLike | None = None, overwrite: bool = False, output_format: str = 'zarr', year_start: int | None = None, year_end: int | None = None, process_variables: list[str] | None = None, **dask_kwargs) → None`

Write out RDRS files to daily-timestep files.

Parameters

- **project** (*str*)
- **input_folder** (*str or os.PathLike*)
- **output_folder** (*str or os.PathLike*)
- **working_folder** (*str or os.PathLike*)
- **overwrite** (*bool*)
- **output_format** (*{“netcdf”, “zarr”}*)
- **year_start** (*int, optional*)
- **year_end** (*int, optional*)
- **process_variables** (*list of str, optional*)
- ****dask_kwargs**

Returns*None***miranda.convert.ecmwf module**

ECMWF TIGGE Conversion module.

`miranda.convert.ecmwf.tigge_convert`(*source: PathLike | None = None, target: PathLike | None = None, processes: int = 8*) → *None*

Convert grib2 file to netCDF format.

Parameters

- **source** (*os.PathLike, optional*)
- **target** (*os.PathLike, optional*)
- **processes** (*int*)

Returns*None***miranda.convert.hq module**

Hydro Quebec Weather Station Data Conversion module.

```
miranda.convert.hq.open_csv(path: str | Path, cf_table: dict | None = {'hurs': {'cell_methods': 'time: point',
'comment': 'The relative humidity with respect to liquid water for T> 0 C, and
with respect to ice for T<0 C.', 'frequency': '1h', 'long_name': 'Near-Surface
Relative Humidity', 'out_name': 'hurs', 'standard_name': 'relative_humidity',
'type': 'real', 'units': '%'}, 'prlp': {'cell_methods': 'time: mean', 'comment': 'At
surface; includes precipitation of all forms of water in the liquid phase.',
'frequency': 'day', 'long_name': 'Rainfall Flux', 'out_name': 'prlp',
'standard_name': 'rainfall_flux', 'type': 'real', 'units': 'kg m-2 s-1'}, 'prsn':
{'cell_methods': 'time: mean', 'comment': 'At surface; includes precipitation of
all forms of water in the solid phase.', 'frequency': 'day', 'long_name': 'Snowfall
Flux', 'out_name': 'prsn', 'standard_name': 'snowfall_flux', 'type': 'real', 'units':
'kg m-2 s-1'}, 'sfcWind': {'cell_methods': 'time: point', 'comment': 'Near-surface
(usually, 10 meters) wind speed.', 'frequency': '1h', 'long_name': 'Near-Surface
Wind Speed', 'out_name': 'sfcWind', 'standard_name': 'wind_speed', 'type':
'real', 'units': 'm s-1'}, 'sfcWindAz': {'cell_methods': 'time: point', 'comment':
'Near-surface (usually, 10 meters) direction from which wind originates.',
'frequency': '1h', 'long_name': 'Near-Surface Wind Direction', 'out_name':
'sfcWindAz', 'standard_name': 'wind_direction', 'type': 'real', 'units': 'degree'},
'snd': {'cell_methods': 'time: point', 'comment': 'The thickness of snow.',
'frequency': '1h', 'long_name': 'Snow Depth', 'out_name': 'snd',
'standard_name': 'surface_snow_thickness', 'type': 'real', 'units': 'm'},
'tasmax_1h': {'cell_methods': 'time: maximum', 'comment': 'Maximum
near-surface (usually, 2 meter) air temperature.', 'frequency': '1h', 'long_name':
'Hourly Maximum Near-Surface Air Temperature', 'out_name': 'tasmax',
'standard_name': 'air_temperature', 'type': 'real', 'units': 'K'}, 'tasmax_day':
{'cell_methods': 'time: maximum', 'comment': 'Maximum near-surface (usually,
2 meter) air temperature.', 'frequency': 'day', 'long_name': 'Daily Maximum
Near-Surface Air Temperature', 'out_name': 'tasmax', 'standard_name':
'air_temperature', 'type': 'real', 'units': 'K'}, 'tasmin_1h': {'cell_methods': 'time:
minimum', 'comment': 'Minimum near-surface (usually, 2 meter) air
temperature.', 'frequency': '1h', 'long_name': 'Hourly Minimum Near-Surface
Air Temperature', 'out_name': 'tasmin', 'standard_name': 'air_temperature',
'type': 'real', 'units': 'K'}, 'tasmin_day': {'cell_methods': 'time: minimum',
'comment': 'Minimum near-surface (usually, 2 meter) air temperature.',
'frequency': 'day', 'long_name': 'Daily Minimum Near-Surface Air Temperature',
'out_name': 'tasmin', 'standard_name': 'air_temperature', 'type': 'real', 'units':
'K'}}) → DataArray
```

Extract daily HQ meteo data and convert to `xr.DataArray` with CF-Convention attributes.

miranda.convert.melcc module

MELCC (Québec) Weather Stations data conversion module.

```
miranda.convert.melcc.concat(files: Sequence[str | Path], output_folder: str | Path, overwrite: bool = True)
→ Path
```

Concatenate converted weather station files.

Parameters

- **files** (sequence of `str` or `Path`)
- **output_folder** (`str` or `Path`)
- **overwrite** (`bool`)

Returns*Path*

`miranda.convert.melcc.convert_mdb(database: str | Path, stations: Dataset, definitions: Dataset, output: str | Path, overwrite: bool = True) → dict[tuple[str, str], pathlib.Path]`

Convert microsoft databases of MELCC observation data to xarray objects.

Parameters

- **database** (*str or Path*)
- **stations**
- **definitions**
- **output**
- **overwrite**

Returns*dict[tuple[str, str], Path]*

`miranda.convert.melcc.convert_melcc_obs(metafile: str | Path, folder: str | Path, output: Path | str | None = None, overwrite: bool = True) → dict[tuple[str, str], pathlib.Path]`

Convert MELCC observation data to xarray data objects, returning paths.

Parameters

- **metafile** (*str or Path*)
- **folder** (*str or Path*)
- **output** (*str or Path, optional*)
- **overwrite** (*bool*)

Returns*dict[str, Path]*

`miranda.convert.melcc.convert_snow_table(file: str | Path, output: str | Path)`

Convert snow data given through an Excel file.

This private data is not included in the MDB files.

Parameters

- **file** (*path*) – The excel file with sheets: “Stations”, “Périodes standards” and “Données”
- **output** (*path*) – Folder where to put the netCDF files (one for each of snd, sd and snw).

`miranda.convert.melcc.list_tables(db_file)`

List the tables of an MDB file.

`miranda.convert.melcc.parse_var_code(vcode: str) → dict[str, Any]`

Parse variable code to generate metadata

Parameters**vcode** (*str*)**Returns***dict[str, Any]*

`miranda.convert.melcc.read_definitions(dbfile: str)`

Read variable definition file using mdbtools.

Parameters

dbfile (*str*)

Returns

pandas.DataFrame

`miranda.convert.melcc.read_stations(db_file: str | PathLike) → DataFrame`

Read station file using mdbtools.

Parameters

db_file (*str or os.PathLike*)

Returns

pandas.DataFrame

`miranda.convert.melcc.read_table(db_file: str | PathLike, tab: str | PathLike) → Dataset`

Read a MySQL table into an xarray object.

Parameters

- **db_file** (*str or os.PathLike*)
- **tab** (*str or os.PathLike*)

Returns

xarray.Dataset

miranda.convert.utils module

Conversion Utilities submodule.

`miranda.convert.utils.date_parser(date: str, *, end_of_period: bool = False, output_type: str = 'str',
strptime_format: str = '%Y-%m-%d') → str | Timestamp | NaTType`

Parses datetime objects from a string representation of a date or both a start and end date.

Parameters

- **date** (*str*) – Date to be converted.
- **end_of_period** (*bool*) – If True, the date will be the end of month or year depending on what's most appropriate.
- **output_type** (*{“datetime”, “str”}*) – Desired returned object type.
- **strptime_format** (*str*) – If `output_type==‘str’`, this sets the strptime format.

Returns

pd.Timestamp or str or pd.NaT – Parsed date.

Notes

Adapted from code written by Gabriel Rondeau-Genesse (@RondeauG)

`miranda.convert.utils.find_version_hash(file: PathLike | str) → dict`

Check for an existing version hash file and, if one cannot be found, generate one from file.

Parameters

file (*str or os.PathLike*)

Returns

dict

miranda.decode package

`class miranda.decode.Decoder(project: str | None)`

Bases: `object`

decode(*files: PathLike | str | list[str | os.PathLike] | generator, chunks: int | None = None, raise_error: bool = False*) → `None`

Decode facets from file or list of files.

Parameters

- **files** (*str or Path or list of str or Path or generator*)
- **chunks** (*int, optional*) – The chunk size used when processing files. Not to be confused with xarray chunks for dimensions.
- **raise_error** (*bool*)

static decode_ahccd_obs(*self, file: PathLike | str*) → `dict`

classmethod decode_cmip5(*file: PathLike | str*) → `dict`

classmethod decode_cmip6(*file: PathLike | str*) → `dict`

classmethod decode_converted(*file: PathLike | str*) → `dict`

classmethod decode_cordex(*file: PathLike | str*) → `dict`

static decode_eccc_obs(*self, file: PathLike | str*) → `dict`

classmethod decode_espo_g6_e51(*file: PathLike | str*) → `dict`

classmethod decode_espo_g6_r2(*file: PathLike | str*) → `dict`

classmethod decode_isimip_ft(*file: PathLike | str*) → `dict`

static decode_melcc_obs(*self, file: PathLike | str*) → `dict`

classmethod decode_nex_gddp_cmip6(*file: PathLike | str*) → `dict`

classmethod decode_pcic_candcs_u6(*file: PathLike | str*) → `dict`

facets_table()

file_facets() → `dict[os.PathLike, dict]`

guess = False

project = None

exception miranda.decode.DecoderError

Bases: Exception

miranda.decode.guess_project(file: PathLike | str) → str

Guess the name of the project

Parameters

file (str or os.PathLike)

Returns

str

Submodules

miranda.decode._decoder module

class miranda.decode._decoder.Decoder(project: str | None)

Bases: object

decode(files: PathLike | str | list[str | os.PathLike] | generator, chunks: int | None = None, raise_error: bool = False) → None

Decode facets from file or list of files.

Parameters

- **files** (str or Path or list of str or Path or generator)
- **chunks** (int, optional) – The chunk size used when processing files. Not to be confused with xarray chunks for dimensions.
- **raise_error** (bool)

static decode_ahccd_obs(self, file: PathLike | str) → dict

classmethod decode_cmip5(file: PathLike | str) → dict

classmethod decode_cmip6(file: PathLike | str) → dict

classmethod decode_converted(file: PathLike | str) → dict

classmethod decode_cordex(file: PathLike | str) → dict

static decode_eccc_obs(self, file: PathLike | str) → dict

classmethod decode_espo_g6_e51(file: PathLike | str) → dict

classmethod decode_espo_g6_r2(file: PathLike | str) → dict

classmethod decode_isimip_ft(file: PathLike | str) → dict

static decode_melcc_obs(self, file: PathLike | str) → dict

classmethod decode_nex_gddp_cmip6(file: PathLike | str) → dict

classmethod decode_pcic_candcs_u6(file: PathLike | str) → dict

facets_table()

file_facets() → dict[os.PathLike, dict]

guess = False

project = None

miranda.decode._decoder.guess_project(*file: PathLike | str*) → str

Guess the name of the project

Parameters

file (*str or os.PathLike*)

Returns

str

miranda.decode._time module

exception miranda.decode._time.DecoderError

Bases: Exception

miranda.eccc package

Environment and Climate Change Canada specialized conversion module.

miranda.eccc.aggregate_stations(*source_files: str | PathLike | None = None, output_folder: str | PathLike | None = None, time_step: str | None = None, variables: str | int | list[str | int] | None = None, include_flags: bool = True, groupings: int | None = None, mf_dataset_freq: str | None = None, temp_directory: str | PathLike | None = None, n_workers: int = 1*) → None

Parameters

- **source_files** (*str or Path*)
- **output_folder** (*str or Path*)
- **variables** (*str or int or list of str or int, optional*)
- **time_step** (*{“hourly”, “daily”}*)
- **include_flags** (*bool*)
- **groupings** (*int*) – The number of files in each group used for converting to multi-file Datasets.
- **mf_dataset_freq** (*str, optional*) – Resampling frequency for creating output multi-file Datasets. E.g. ‘YS’: 1 year per file, ‘5YS’: 5 years per file.
- **temp_directory** (*str or Path, optional*) – Use another temporary directory location in case default location is not spacious enough.
- **n_workers** (*int*)

Returns

None

`miranda.eccc.convert_ahccd(data_source: str | Path, output_dir: str | Path, variable: str, generation: int | None = None) → None`

Convert Adjusted and Homogenized Canadian Climate Dataset files.

Parameters

- **data_source** (*str or Path*)
- **output_dir** (*str or Path*)
- **variable** (*str*)
- **generation** (*int, optional*)

Returns

None

`miranda.eccc.convert_ahccd_fwf_files(ff: Path | str, metadata: DataFrame, variable: str, generation: int | None = None, cols_specs: list[tuple[int, int]] | None = None, attrs: dict | None = None) → Dataset`

Convert AHCCD fixed-width files.

Parameters

- **ff** (*str or Path*)
- **metadata** (*pandas.DataFrame*)
- **variable** (*str*)
- **generation**
- **cols_specs**
- **attrs**

Returns

xarray.Dataset

`miranda.eccc.convert_flat_files(source_files: str | PathLike, output_folder: str | PathLike | list[str | int], variables: str | int | list[str | int], mode: str = 'hourly', n_workers: int = 4) → None`

Parameters

- **source_files** (*str or Path*)
- **output_folder** (*str or Path*)
- **variables** (*str or List[str]*)
- **mode** (*{“hourly”, “daily”}*)
- **n_workers** (*int*)

Returns

None

`miranda.eccc.daily_summaries_to_netcdf(station: dict, path_output: Path | str) → None`

Convert daily climate summaries to NetCDF files.

Uses xarray to transform the ‘station’ from `find_and_extract_dly` into a CF-Convention netCDF file

Parameters

- **station** (*dict*) – dict created by using `find_and_extract_dly`

- **path_output** (*str or Path*) – Output path.

Returns*None*

`miranda.eccc.extract_daily_summaries(path_station: Path | str, rm_flags: bool = False, file_suffix: str = 'csv') → dict`

Extract daily climate summaries from ECCC CSV files.

Parameters

- **path_station** (*str or Path*) – PathLike or *str* to the station’s folder containing the csv files.
- **rm_flags** (*bool*) – Removes the ‘Flag’ and ‘Quality’ columns of the ECCC files.
- **file_suffix** (*str*) – File suffixes used by the tabular data. Default: “.csv”.

Returns

dict – dict containing the station metadata, as well as the data stored within a pandas Dataframe.

`miranda.eccc.merge_converted_variables(source_files: str | PathLike, output_folder: str | PathLike, variables: str | int | list[str | int] | None = None, station_metadata: str | PathLike | None = None, overwrite: bool = False, n_workers: int = 1) → None`

Parameters

- **source_files** (*str, Path*)
- **output_folder** (*str, Path*)
- **variables** (*str or int or list of str or int, optional*)
- **station_metadata** (*str or Path, optional*)
- **overwrite** (*bool*)
- **n_workers** (*int*)

Returns*None***Submodules****miranda.eccc._homogenized module**

Adjusted and Homogenized Canadian Clime Data module.

`miranda.eccc._homogenized.convert_ahccd(data_source: str | Path, output_dir: str | Path, variable: str, generation: int | None = None) → None`

Convert Adjusted and Homogenized Canadian Climate Dataset files.

Parameters

- **data_source** (*str or Path*)
- **output_dir** (*str or Path*)
- **variable** (*str*)
- **generation** (*int, optional*)

Returns

None

`miranda.eccc._homogenized.convert_ahccd_fw_files(ff: Path | str, metadata: DataFrame, variable: str, generation: int | None = None, cols_specs: list[tuple[int, int]] | None = None, attrs: dict | None = None) → Dataset`

Convert AHCCD fixed-width files.

Parameters

- **ff** (*str or Path*)
- **metadata** (*pandas.DataFrame*)
- **variable** (*str*)
- **generation**
- **cols_specs**
- **attrs**

Returns

xarray.Dataset

miranda.eccc._raw module

`miranda.eccc._raw.aggregate_stations(source_files: str | PathLike | None = None, output_folder: str | PathLike | None = None, time_step: str | None = None, variables: str | int | list[str | int] | None = None, include_flags: bool = True, groupings: int | None = None, mf_dataset_freq: str | None = None, temp_directory: str | PathLike | None = None, n_workers: int = 1) → None`

Parameters

- **source_files** (*str or Path*)
- **output_folder** (*str or Path*)
- **variables** (*str or int or list of str or int, optional*)
- **time_step** (*{“hourly”, “daily”}*)
- **include_flags** (*bool*)
- **groupings** (*int*) – The number of files in each group used for converting to multi-file Datasets.
- **mf_dataset_freq** (*str, optional*) – Resampling frequency for creating output multi-file Datasets. E.g. ‘YS’: 1 year per file, ‘5YS’: 5 years per file.
- **temp_directory** (*str or Path, optional*) – Use another temporary directory location in case default location is not spacious enough.
- **n_workers** (*int*)

Returns

None

`miranda.eccc._raw.convert_flat_files`(*source_files*: *str* | *PathLike*, *output_folder*: *str* | *PathLike* | *list*[*str* | *int*], *variables*: *str* | *int* | *list*[*str* | *int*], *mode*: *str* = 'hourly', *n_workers*: *int* = 4) → *None*

Parameters

- **source_files** (*str* or *Path*)
- **output_folder** (*str* or *Path*)
- **variables** (*str* or *List*[*str*])
- **mode** ({ "hourly", "daily" })
- **n_workers** (*int*)

Returns

None

`miranda.eccc._raw.merge_converted_variables`(*source_files*: *str* | *PathLike*, *output_folder*: *str* | *PathLike*, *variables*: *str* | *int* | *list*[*str* | *int*] | *None* = *None*, *station_metadata*: *str* | *PathLike* | *None* = *None*, *overwrite*: *bool* = *False*, *n_workers*: *int* = 1) → *None*

Parameters

- **source_files** (*str*, *Path*)
- **output_folder** (*str*, *Path*)
- **variables** (*str* or *int* or *list* of *str* or *int*, optional)
- **station_metadata** (*str* or *Path*, optional)
- **overwrite** (*bool*)
- **n_workers** (*int*)

Returns

None

miranda.eccc._summaries module

`miranda.eccc._summaries.daily_summaries_to_netcdf`(*station*: *dict*, *path_output*: *Path* | *str*) → *None*

Convert daily climate summaries to NetCDF files.

Uses xarray to transform the 'station' from `find_and_extract_dly` into a CF-Convention netCDF file

Parameters

- **station** (*dict*) – dict created by using `find_and_extract_dly`
- **path_output** (*str* or *Path*) – Output path.

Returns

None

`miranda.eccc._summaries.extract_daily_summaries`(*path_station*: *Path* | *str*, *rm_flags*: *bool* = *False*, *file_suffix*: *str* = '.csv') → *dict*

Extract daily climate summaries from ECCC CSV files.

Parameters

- **path_station** (*str* or *Path*) – *PathLike* or *str* to the station's folder containing the csv files.

- **rm_flags** (*bool*) – Removes the ‘Flag’ and ‘Quality’ columns of the ECCC files.
- **file_suffix** (*str*) – File suffixes used by the tabular data. Default: “.csv”.

Returns

dict – dict containing the station metadata, as well as the data stored within a pandas Dataframe.

miranda.eccc._support_rvt module

class miranda.eccc._support_rvt.**Path**(*args, **kwargs)

Bases: PurePath

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

absolute()

Return an absolute version of this path. This function works even if the path doesn’t point to anything.

No normalization is done, i.e. all ‘.’ and ‘..’ will be kept along. Use resolve() to get the canonical path to a file.

chmod(*mode*, *, *follow_symlinks=True*)

Change the permissions of the path, like os.chmod().

classmethod cwd()

Return a new path pointing to the current working directory (as returned by os.getcwd()).

exists()

Whether this path exists.

expanduser()

Return a new path with expanded ~ and ~user constructs (as returned by os.path.expanduser)

glob(*pattern*)

Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given relative pattern.

group()

Return the group name of the file gid.

hardlink_to(*target*)

Make this path a hard link pointing to the same file as *target*.

Note the order of arguments (self, target) is the reverse of os.link’s.

classmethod home()

Return a new path pointing to the user’s home directory (as returned by os.path.expanduser('~')).

is_block_device()

Whether this path is a block device.

is_char_device()

Whether this path is a character device.

is_dir()

Whether this path is a directory.

is_fifo()

Whether this path is a FIFO.

is_file()

Whether this path is a regular file (also True for symlinks pointing to regular files).

is_mount()

Check if this path is a POSIX mount point

is_socket()

Whether this path is a socket.

is_symlink()

Whether this path is a symbolic link.

iterdir()

Iterate over the files in this directory. Does not yield any result for the special paths '.' and '..'.

lchmod(mode)

Like chmod(), except if the path points to a symlink, the symlink's permissions are changed, rather than its target's.

link_to(target)

Make the target path a hard link pointing to this path.

Note this function does not make this path a hard link to *target*, despite the implication of the function and argument names. The order of arguments (target, link) is the reverse of Path.symlink_to, but matches that of os.link.

Deprecated since Python 3.10 and scheduled for removal in Python 3.12. Use *hardlink_to()* instead.

lstat()

Like stat(), except if the path points to a symlink, the symlink's status information is returned, rather than its target's.

mkdir(mode=511, parents=False, exist_ok=False)

Create a new directory at this given path.

open(mode='r', buffering=-1, encoding=None, errors=None, newline=None)

Open the file pointed by this path and return a file object, as the built-in open() function does.

owner()

Return the login name of the file owner.

read_bytes()

Open the file in bytes mode, read it, and close the file.

read_text(encoding=None, errors=None)

Open the file in text mode, read it, and close the file.

readlink()

Return the path to which the symbolic link points.

rename(*target*)

Rename this path to the target path.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

replace(*target*)

Rename this path to the target path, overwriting if that path exists.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

resolve(*strict=False*)

Make the path absolute, resolving all symlinks on the way and also normalizing it (for example turning slashes into backslashes under Windows).

rglob(*pattern*)

Recursively yield all existing files (of any kind, including directories) matching the given relative pattern, anywhere in this subtree.

rmdir()

Remove this directory. The directory must be empty.

samefile(*other_path*)

Return whether *other_path* is the same or not as this file (as returned by `os.path.samefile()`).

stat(**, follow_symlinks=True*)

Return the result of the `stat()` system call on this path, like `os.stat()` does.

symlink_to(*target, target_is_directory=False*)

Make this path a symlink pointing to the target path. Note the order of arguments (link, target) is the reverse of `os.symlink`.

touch(*mode=438, exist_ok=True*)

Create this file with the given access mode, if it doesn't exist.

unlink(*missing_ok=False*)

Remove this file or link. If the path is a directory, use `rmdir()` instead.

write_bytes(*data*)

Open the file in bytes mode, write to it, and close the file.

write_text(*data, encoding=None, errors=None, newline=None*)

Open the file in text mode, write to it, and close the file.

`miranda.eccc._support_rvt.gather_eccc_stations`(*timestep: str, start_date: datetime | str | None = None, end_date: datetime | str | None = None, climate_id: str | None = None*) → `DataFrame`

Collect ECCC station data from the Environment and Climate Change Canada API.

Parameters

- **timestep** (*str*)
- **start_date** (*datetime.datetime or str, optional*)
- **end_date** (*datetime.datetime or str, optional*)

- **climate_id** (*str, optional*)

Returns

pandas.DataFrame

miranda.eccc._utils module

`miranda.eccc._utils.cf_ahccd_metadata(code: str, gen: int)`

Parameters

- **code** (*{“dx”, “dn”, “dm”, “dt”, “ds”, “dr”}*)
- **gen** (*{1, 2, 3}*)

Returns

dict[str, int or str or float], dict, list[tuple[int, int]], int

`miranda.eccc._utils.cf_station_metadata(variable_code: int | str) → Mapping[str, int | float | str]`

Parameters

variable_code (*int or str*)

Returns

dict

miranda.ecmwf package

ECMWF Download module.

`miranda.ecmwf.rename_era5_files(path: PathLike | str) → None`

Rename badly named ERA5 files.

Notes

Requires that the proper ERA5 project name is in the filename, separated by underscores. Assumes that the data

Parameters

path (*os.PathLike or str*) – Path to a folder containing netcdf files

Returns

None

`miranda.ecmwf.request_era5(projects: str | list[str], *, variables: str | Sequence[str] | None = None, domain: str = 'AMNO', pressure_levels: list[int] | None = None, separate_pressure_levels: bool = True, output_folder: str | PathLike | None = None, year_start: str | int | None = None, year_end: str | int | None = None, dry_run: bool = False, processes: int = 10, url: str | None = None, key: str | None = None) → None`

Request ERA5/ERA5-Land from Copernicus Data Store in NetCDF4 format.

Parameters

- **projects** (*str or List[str]*) – Allowed keys: {“era5-land”, “era5-land-monthly-means”, “era5-single-levels”, “era5-single-levels-monthly-means”, “era5-single-levels-preliminary-back-extension”, “era5-single-levels-monthly-means-preliminary-back-extension”, “era5-pressure-levels”, “era5-pressure-levels-monthly-means”, “era5-pressure-levels-preliminary-back-extension”, “era5-pressure-levels-monthly-means-preliminary-back-extension”}

- **variables** (*str or Sequence[str]*) – Variable codes requested. If None, will attempt all hard-coded variables supported by miranda converter.
- **domain** (*{“GLOBAL”, “AMNO”, “NAM”, “CAN”, “QC”, “MTL”}*) – Geographic domain requested. Default: “AMNO” (North America).
- **pressure_levels** (*List[int], optional*) – If set and project requested has pressure levels, will download specific pressure levels.
- **separate_pressure_levels** (*bool*) – Whether to separate files for each pressure level. Default: True
- **output_folder** (*str or os.PathLike, optional*) – Folder to send files to. If None, will create a “downloaded” folder in current working directory.
- **year_start** (*int, optional*) – Starting year for data download. If None, will download from first available year for project.
- **year_end** (*int, optional*) – End year for data download. If None, will download files for current year and two months prior to present day.
- **dry_run** (*bool*) – Do not send request. For debugging purposes.
- **processes** (*int*) – The number of simultaneous download requests. Default: 10.
- **url** (*str, optional*) – URL for Copernicus Data Store API (if not already using .cdsapirc)
- **key** (*str, optional*) – Personal access key for Copernicus Data Store (if not already using .cdsapirc)

Returns

None

```
miranda.ecmwf.request_tigge(variables: list[str], providers: list[str] | None = None, *, forecast_type: str =
                             'pf', times: list[str] | None = None, dates: list[str] | None = None, date_start: str
                             | None = None, date_end: str | None = None, output_folder: PathLike | None =
                             None, processes: int = 4) → None
```

Request tigge data from ECMWF in grib format.

Parameters

- **variables** (*list of str*)
- **providers** (*list of str, optional*)
- **forecast_type** (*{“pf”, “cf”}*)
- **times** (*list of str, optional*)
- **dates** (*list of str, optional*)
- **date_start** (*str, optional*)
- **date_end** (*str, optional*)
- **output_folder** (*os.PathLike, optional*)
- **processes** (*int*)

Returns

None

Submodules

miranda.ecmwf._era5 module

`miranda.ecmwf._era5.rename_era5_files(path: PathLike | str) → None`

Rename badly named ERA5 files.

Notes

Requires that the proper ERA5 project name is in the filename, separated by underscores. Assumes that the data

Parameters

path (*os.PathLike or str*) – Path to a folder containing netcdf files

Returns

None

`miranda.ecmwf._era5.request_era5(projects: str | list[str], *, variables: str | Sequence[str] | None = None, domain: str = 'AMNO', pressure_levels: list[int] | None = None, separate_pressure_levels: bool = True, output_folder: str | PathLike | None = None, year_start: str | int | None = None, year_end: str | int | None = None, dry_run: bool = False, processes: int = 10, url: str | None = None, key: str | None = None) → None`

Request ERA5/ERA5-Land from Copernicus Data Store in NetCDF4 format.

Parameters

- **projects** (*str or List[str]*) – Allowed keys: {"era5-land", "era5-land-monthly-means", "era5-single-levels", "era5-single-levels-monthly-means", "era5-single-levels-preliminary-back-extension", "era5-single-levels-monthly-means-preliminary-back-extension", "era5-pressure-levels", "era5-pressure-levels-monthly-means", "era5-pressure-levels-preliminary-back-extension", "era5-pressure-levels-monthly-means-preliminary-back-extension"}
- **variables** (*str or Sequence[str]*) – Variable codes requested. If None, will attempt all hard-coded variables supported by miranda converter.
- **domain** (*{ "GLOBAL", "AMNO", "NAM", "CAN", "QC", "MTL" }*) – Geographic domain requested. Default: "AMNO" (North America).
- **pressure_levels** (*List[int], optional*) – If set and project requested has pressure levels, will download specific pressure levels.
- **separate_pressure_levels** (*bool*) – Whether to separate files for each pressure level. Default: True
- **output_folder** (*str or os.PathLike, optional*) – Folder to send files to. If None, will create a "downloaded" folder in current working directory.
- **year_start** (*int, optional*) – Starting year for data download. If None, will download from first available year for project.
- **year_end** (*int, optional*) – End year for data download. If None, will download files for current year and two months prior to present day.
- **dry_run** (*bool*) – Do not send request. For debugging purposes.
- **processes** (*int*) – The number of simultaneous download requests. Default: 10.
- **url** (*str, optional*) – URL for Copernicus Data Store API (if not already using .cdsapirc)

- **key** (*str, optional*) – Personal access key for Copernicus Data Store (if not already using .cdsapirc)

Returns

None

miranda.ecmwf._tigge module

`miranda.ecmwf._tigge.request_tigge(variables: list[str], providers: list[str] | None = None, *, forecast_type: str = 'pf', times: list[str] | None = None, dates: list[str] | None = None, date_start: str | None = None, date_end: str | None = None, output_folder: PathLike | None = None, processes: int = 4) → None`

Request tigge data from ECMWF in grib format.

Parameters

- **variables** (*list of str*)
- **providers** (*list of str, optional*)
- **forecast_type** (*{“pf”, “cf”}*)
- **times** (*list of str, optional*)
- **dates** (*list of str, optional*)
- **date_start** (*str, optional*)
- **date_end** (*str, optional*)
- **output_folder** (*os.PathLike, optional*)
- **processes** (*int*)

Returns

None

miranda.gis package

GIS Operations module.

`miranda.gis.add_ar6_regions(ds: Dataset) → Dataset`

Add the IPCC AR6 Regions to dataset.

Parameters

ds (*xarray.Dataset*)

Returns

xarray.Dataset

`miranda.gis.subset_domain(ds: Dataset | DataArray, domain: str, **kwargs) → Dataset | DataArray`

Subset an xarray object according to a specific domain.

Notes

Requires installation of GIS libraries.

Parameters

- **ds** (*xarray.Dataset or xarray.DataArray*)
- **domain** (*str*)
- ****kwargs**

Returns

xarray.Dataset or xarray.DataArray

`miranda.gis.subsetting_domains(domain: str) → list`

Provides the bounding box coordinates for specific domains.

Parameters

domain (*{“global”, “nam”, “can”, “qc”, “mtl”}*)

Returns

np.array – North, West, South, and East coordinates

Submodules

miranda.gis._domains module

`miranda.gis._domains.add_ar6_regions(ds: Dataset) → Dataset`

Add the IPCC AR6 Regions to dataset.

Parameters

ds (*xarray.Dataset*)

Returns

xarray.Dataset

`miranda.gis._domains.subset_domain(ds: Dataset | DataArray, domain: str, **kwargs) → Dataset | DataArray`

Subset an xarray object according to a specific domain.

Notes

Requires installation of GIS libraries.

Parameters

- **ds** (*xarray.Dataset or xarray.DataArray*)
- **domain** (*str*)
- ****kwargs**

Returns

xarray.Dataset or xarray.DataArray

`miranda.gis._domains.subsetting_domains(domain: str) → list`

Provides the bounding box coordinates for specific domains.

Parameters

domain ({*“global”*, *“nam”*, *“can”*, *“qc”*, *“mtl”*})

Returns

np.array – North, West, South, and East coordinates

miranda.io package

IO Utilities module.

`miranda.io.concat_rechunk_zarr(freq: str, input_folder: str | PathLike, output_folder: str | PathLike, overwrite: bool = False, **dask_kwargs) → None`

Concatenate and rechunk zarr files.

Parameters

- **freq** (*str*)
- **input_folder** (*str* or *os.PathLike*)
- **output_folder** (*str* or *os.PathLike*)
- **overwrite** (*bool*)
- ****dask_kwargs**

Returns

None

`miranda.io.discover_data(input_files: str | PathLike | list[str | os.PathLike] | generator, suffix: str = 'nc', recurse: bool = True) → list[pathlib.Path] | generator`

Discover data.

Parameters

- **input_files** (*str*, *pathlib.Path*, *list of str or Path*, or *GeneratorType*) – Path or string to a file, a folder, or a generator of paths.
- **suffix** (*str*) – File-ending suffix to search for. Default: *“nc”*.
- **recurse** (*bool*) – Whether to recurse through folders or not. Default: *True*.

Returns

list of pathlib.Path or GeneratorType of pathlib.Path

Warning: Recursion through *“zarr”* files is explicitly disabled. Recursive globs and generators will not be expanded/sorted.

```
miranda.io.fetch_chunk_config(priority: str, freq: str, dims: Sequence[str] | dict[str, int] | Frozen |
                             tuple[collections.abc.Hashable], default_config: dict = {'files': {'1hr':
{'default': {'lat': 250, 'lon': 250, 'time': 168}, 'rotated': {'rlat': 250, 'rlon':
250, 'time': 168}}, 'day': {'default': {'lat': 125, 'lon': 125, 'time': '1 year'},
'rotated': {'rlat': 125, 'rlon': 125, 'time': '1 year'}}, 'month': {'default': {'lat':
500, 'lon': 500, 'time': 120}, 'rotated': {'rlat': 500, 'rlon': 500, 'time': 120}}},
'time': {'1hr': {'default': {'lat': 50, 'lon': 50, 'time': 1440}, 'rotated': {'rlat':
50, 'rlon': 50, 'time': 1440}}, 'day': {'default': {'lat': 50, 'lon': 50, 'time': '4
years'}, 'rotated': {'rlat': 50, 'rlon': 50, 'time': '4 years'}}, 'month': {'default':
{'lat': 250, 'lon': 250, 'time': 240}, 'rotated': {'rlat': 250, 'rlon': 250, 'time':
240}}})) → dict[str, int]
```

Parameters

- **priority** ({*“time”*, *“files”*}) – Specifies whether the chunking regime should prioritize file granularity (*“files”*) or time series (*“time”*).
- **freq** ({*“1hr”*, *“day”*, *“month”*}) – The time frequency of the input data.
- **dims** (*sequence of str*) – The dimension names that will be used for chunking.
- **default_config** (*dict*) – The dictionary to use for determining the chunking configuration.

Returns

dict[str, int]

```
miranda.io.find_filepaths(source: str | Path | generator | list[pathlib.Path | str], recursive: bool = True,
                          file_suffixes: str | list[str] | None = None, **_) → list[pathlib.Path]
```

Find all available filepaths at a given source.

Parameters

- **source** (*str, Path, GeneratorType, or list[str or Path]*)
- **recursive** (*bool*)
- **file_suffixes** (*str or list of str, optional*)

Returns

list of pathlib.Path

```
miranda.io.merge_rechunk_zarrs(input_folder: str | PathLike, output_folder: str | PathLike, project: str | None
                               = None, target_chunks: dict[str, int] | None = None, variables:
                               Sequence[str] | None = None, freq: str | None = None, suffix: str = 'zarr',
                               overwrite: bool = False) → None
```

Merge and rechunk zarr files.

Parameters

- **input_folder** (*str or os.PathLike*)
- **output_folder** (*str or os.PathLike*)
- **project** (*str, optional*)
- **target_chunks** (*dict[str, int], optional*)
- **variables** (*Sequence of str, optional*)
- **freq** (*str, optional*)
- **suffix** ({*“nc”*, *“zarr”*})
- **overwrite** (*bool*)

Returns

None

`miranda.io.prepare_chunks_for_ds(ds: Dataset, chunks: dict[str, str | int]) → dict[str, int]`

Prepare the chunks to be used to write Dataset.

This includes translating the time chunks, making sure chunks are not too small, and removing -1.

Parameters

- **ds** (*xr.Dataset*) – Dataset that we want to write with the chunks.
- **chunks** (*dict*) – Desired chunks in human-readable format (with “4 years” and -1).

Returns

dict – Chunks in a format that is ready to be used to write to disk.

`miranda.io.rechunk_files(input_folder: str | PathLike, output_folder: str | PathLike, project: str | None = None, time_step: str | None = None, chunking_priority: str = 'auto', target_chunks: dict[str, int] | None = None, variables: Sequence[str] | None = None, suffix: str = 'nc', output_format: str = 'netcdf', overwrite: bool = False) → None`

Rechunks dataset for better loading/reading performance.

Warning: Globbing assumes that target datasets to be rechunked have been saved in NetCDF format. File naming requires the following order of facets: `{variable}_{time_step}_{institute}_{project}_reanalysis_*.nc`. Chunking dimensions are assumed to be CF-Compliant (*lat, lon, rlat, rlon, time*).

Parameters

- **input_folder** (*str or os.PathLike*) – Folder to be examined. Performs globbing.
- **output_folder** (*str or os.PathLike*) – Target folder.
- **project** (*str, optional*) – Supported projects. Used for determining chunk dictionary. Superseded if *target_chunks* is set.
- **time_step** (*{“1hr”, “day”}, optional*) – Time step of the input data. Parsed from dataset attrs if not set. Superseded if *target_chunks* is set.
- **chunking_priority** (*{“time”, “files”, “auto”}*) – The chunking regime to use. Default: “auto”.
- **target_chunks** (*dict, optional*) – Must include “time”, optionally “lat” and “lon”, depending on dataset structure.
- **variables** (*Sequence[str], optional*) – If no variables set, will attempt to process all variables supported based on project name.
- **suffix** (*{“nc”, “zarr”}*) – Suffix used to identify data files. Default: “nc”.
- **output_format** (*{“netcdf”, “zarr”}*) – Default: “zarr”.
- **overwrite** (*bool*) – Will overwrite files. For zarr, existing folders will be removed before writing.

Returns

None

`miranda.io.translate_time_chunk(chunks: dict, calendar: str, timesize: int) → dict`

Translate chunk specification for time into a number.

Notes

-1 translates to *timesize* ‘Nyear’ translates to N times the number of days in a year of calendar *calendar*.

`miranda.io.write_dataset(ds: dataArray | Dataset, output_path: str | PathLike, output_format: str, chunks: dict | None = None, overwrite: bool = False, compute: bool = True) → dict[str, pathlib.Path]`

Write xarray object to NetCDF or Zarr with appropriate chunking regime.

Parameters

- **ds** (*xr.DataArray or xr.Dataset*) – Dataset or DataArray.
- **output_path** (*str or os.PathLike*) – Output folder path.
- **output_format** (*{“netcdf”, “zarr”}*) – Output data container type.
- **chunks** (*dict, optional*) – Chunking layout to be written to new files. If None, chunking will be left to the relevant backend engine.
- **overwrite** (*bool*) – Whether to remove existing files or fail if files already exist.
- **compute** (*bool*) – If True, files will be converted with each call to file conversion. If False, will return a `dask.Delayed` object that can be computed later. Default: True.

Returns

dict[str, Path]

`miranda.io.write_dataset_dict(dataset_dict: dict[str, xarray.core.dataset.Dataset | None], output_folder: str | PathLike, temp_folder: str | PathLike, *, output_format: str = 'zarr', overwrite: bool = False, chunks: dict[str, int], **dask_kwargs)`

Write dataset from Miranda-formatted dataset.

Parameters

- **dataset_dict** (*dict[str, xr.Dataset or None]*)
- **output_folder** (*str or os.PathLike*)
- **temp_folder** (*str or os.PathLike*)
- **output_format** (*{“netcdf”, “zarr”}*)
- **overwrite** (*bool*)
- **chunks** (*dict[str, int]*)
- ****dask_kwargs**

Returns

None

Submodules

miranda.io._input module

`miranda.io._input.discover_data(input_files: str | PathLike | list[str | os.PathLike] | generator, suffix: str = 'nc', recurse: bool = True) → list[pathlib.Path] | generator`

Discover data.

Parameters

- **input_files** (*str*, *pathlib.Path*, *list of str or Path*, or *GeneratorType*) – Path or string to a file, a folder, or a generator of paths.
- **suffix** (*str*) – File-ending suffix to search for. Default: “nc”.
- **recurse** (*bool*) – Whether to recurse through folders or not. Default: True.

Returns

list of pathlib.Path or GeneratorType of pathlib.Path

Warning: Recursion through “.zarr” files is explicitly disabled. Recursive globs and generators will not be expanded/sorted.

`miranda.io._input.find_filepaths(source: str | Path | generator | list[pathlib.Path | str], recursive: bool = True, file_suffixes: str | list[str] | None = None, **_) → list[pathlib.Path]`

Find all available filepaths at a given source.

Parameters

- **source** (*str*, *Path*, *GeneratorType*, or *list[str or Path]*)
- **recursive** (*bool*)
- **file_suffixes** (*str or list of str, optional*)

Returns

list of pathlib.Path

miranda.io._output module

IO Output Operations module.

`miranda.io._output.concat_rechunk_zarr(freq: str, input_folder: str | PathLike, output_folder: str | PathLike, overwrite: bool = False, **dask_kwargs) → None`

Concatenate and rechunk zarr files.

Parameters

- **freq** (*str*)
- **input_folder** (*str or os.PathLike*)
- **output_folder** (*str or os.PathLike*)
- **overwrite** (*bool*)
- ****dask_kwargs**

Returns

None

`miranda.io._output.merge_rechunk_zarrs(input_folder: str | PathLike, output_folder: str | PathLike, project: str | None = None, target_chunks: dict[str, int] | None = None, variables: Sequence[str] | None = None, freq: str | None = None, suffix: str = 'zarr', overwrite: bool = False) → None`

Merge and rechunk zarr files.

Parameters

- **input_folder** (*str or os.PathLike*)

- **output_folder** (*str or os.PathLike*)
- **project** (*str, optional*)
- **target_chunks** (*dict[str, int], optional*)
- **variables** (*Sequence of str, optional*)
- **freq** (*str, optional*)
- **suffix** (*{“nc”, “zarr”}*)
- **overwrite** (*bool*)

Returns*None*

`miranda.io._output.write_dataset(ds: DataArray | Dataset, output_path: str | PathLike, output_format: str, chunks: dict | None = None, overwrite: bool = False, compute: bool = True) → dict[str, pathlib.Path]`

Write xarray object to NetCDF or Zarr with appropriate chunking regime.

Parameters

- **ds** (*xr.DataArray or xr.Dataset*) – Dataset or DataArray.
- **output_path** (*str or os.PathLike*) – Output folder path.
- **output_format** (*{“netcdf”, “zarr”}*) – Output data container type.
- **chunks** (*dict, optional*) – Chunking layout to be written to new files. If None, chunking will be left to the relevant backend engine.
- **overwrite** (*bool*) – Whether to remove existing files or fail if files already exist.
- **compute** (*bool*) – If True, files will be converted with each call to file conversion. If False, will return a `dask.Delayed` object that can be computed later. Default: True.

Returns*dict[str, Path]*

`miranda.io._output.write_dataset_dict(dataset_dict: dict[str, xr.Dataset or None], output_folder: str | PathLike, temp_folder: str | PathLike, *, output_format: str = 'zarr', overwrite: bool = False, chunks: dict[str, int], **dask_kwargs)`

Write dataset from Miranda-formatted dataset.

Parameters

- **dataset_dict** (*dict[str, xr.Dataset or None]*)
- **output_folder** (*str or os.PathLike*)
- **temp_folder** (*str or os.PathLike*)
- **output_format** (*{“netcdf”, “zarr”}*)
- **overwrite** (*bool*)
- **chunks** (*dict[str, int]*)
- ****dask_kwargs**

Returns*None*

miranda.io._rechunk module

`miranda.io._rechunk.fetch_chunk_config`(*priority: str, freq: str, dims: Sequence[str] | dict[str, int] | Frozen | tuple[collections.abc.Hashable]*, *default_config: dict = {'files': {'1hr': {'default': {'lat': 250, 'lon': 250, 'time': 168}, 'rotated': {'rlat': 250, 'rlon': 250, 'time': 168}}, 'day': {'default': {'lat': 125, 'lon': 125, 'time': '1 year'}, 'rotated': {'rlat': 125, 'rlon': 125, 'time': '1 year'}}, 'month': {'default': {'lat': 500, 'lon': 500, 'time': 120}, 'rotated': {'rlat': 500, 'rlon': 500, 'time': 120}}}, 'time': {'1hr': {'default': {'lat': 50, 'lon': 50, 'time': 1440}, 'rotated': {'rlat': 50, 'rlon': 50, 'time': 1440}}, 'day': {'default': {'lat': 50, 'lon': 50, 'time': '4 years'}, 'rotated': {'rlat': 50, 'rlon': 50, 'time': '4 years'}}, 'month': {'default': {'lat': 250, 'lon': 250, 'time': 240}, 'rotated': {'rlat': 250, 'rlon': 250, 'time': 240}}})*) → dict[str, int]

Parameters

- **priority** (*{“time”, “files”}*) – Specifies whether the chunking regime should prioritize file granularity (“files”) or time series (“time”).
- **freq** (*{“1hr”, “day”, “month”}*) – The time frequency of the input data.
- **dims** (*sequence of str*) – The dimension names that will be used for chunking.
- **default_config** (*dict*) – The dictionary to use for determining the chunking configuration.

Returns

dict[str, int]

`miranda.io._rechunk.prepare_chunks_for_ds`(*ds: Dataset, chunks: dict[str, str | int]*) → dict[str, int]

Prepare the chunks to be used to write Dataset.

This includes translating the time chunks, making sure chunks are not too small, and removing -1.

Parameters

- **ds** (*xr.Dataset*) – Dataset that we want to write with the chunks.
- **chunks** (*dict*) – Desired chunks in human-readable format (with “4 years” and -1).

Returns

dict – Chunks in a format that is ready to be used to write to disk.

`miranda.io._rechunk.rechunk_files`(*input_folder: str | PathLike, output_folder: str | PathLike, project: str | None = None, time_step: str | None = None, chunking_priority: str = 'auto', target_chunks: dict[str, int] | None = None, variables: Sequence[str] | None = None, suffix: str = 'nc', output_format: str = 'netcdf', overwrite: bool = False*) → None

Rechunks dataset for better loading/reading performance.

Warning: Globbing assumes that target datasets to be rechunked have been saved in NetCDF format. File naming requires the following order of facets: *{variable}_{time_step}_{institute}_{project}_reanalysis_*.nc*. Chunking dimensions are assumed to be CF-Compliant (*lat, lon, rlat, rlon, time*).

Parameters

- **input_folder** (*str or os.PathLike*) – Folder to be examined. Performs globbing.
- **output_folder** (*str or os.PathLike*) – Target folder.

- **project** (*str, optional*) – Supported projects. Used for determining chunk dictionary. Superseded if *target_chunks* is set.
- **time_step** (*{“1hr”, “day”}, optional*) – Time step of the input data. Parsed from dataset attrs if not set. Superseded if *target_chunks* is set.
- **chunking_priority** (*{“time”, “files”, “auto”}*) – The chunking regime to use. Default: “auto”.
- **target_chunks** (*dict, optional*) – Must include “time”, optionally “lat” and “lon”, depending on dataset structure.
- **variables** (*Sequence[str], optional*) – If no variables set, will attempt to process all variables supported based on project name.
- **suffix** (*{“nc”, “zarr”}*) – Suffix used to identify data files. Default: “nc”.
- **output_format** (*{“netcdf”, “zarr”}*) – Default: “zarr”.
- **overwrite** (*bool*) – Will overwrite files. For zarr, existing folders will be removed before writing.

Returns*None*

`miranda.io._rechunk.translate_time_chunk(chunks: dict, calendar: str, timesize: int) → dict`
 Translate chunk specification for time into a number.

Notes

-1 translates to *timesize* ‘Nyear’ translates to N times the number of days in a year of calendar *calendar*.

miranda.io.utils module

IO Utilities module.

`miranda.io.utils.creation_date(path_to_file: str | PathLike) → float | date`

Return the date that a file was created, falling back to when it was last modified if unable to determine.

See <https://stackoverflow.com/a/39501288/1709587> for explanation.

Parameters

path_to_file (*str or os.PathLike*)

Returns*float or date*

`miranda.io.utils.delayed_write(ds: Dataset, outfile: str | PathLike, output_format: str, overwrite: bool, target_chunks: dict | None = None) → delayed`

Stage a Dataset writing job using *dask.delayed* objects.

Parameters

- **ds** (*xr.Dataset*)
- **outfile** (*str or os.PathLike*)
- **target_chunks** (*dict*)
- **output_format** (*{“netcdf”, “zarr”}*)

- **overwrite** (*bool*)

Returns

dask.delayed.delayed

`miranda.io.utils.get_chunks_on_disk(file: PathLike | str) → dict`

Determine the chunks on disk for a given NetCDF or Zarr file.

Parameters

file (*str or os.PathLike*) – File to be examined. Supports NetCDF and Zarr.

Returns

dict

`miranda.io.utils.get_global_attrs(file_or_dataset: str | PathLike | Dataset) → dict[str, str | int]`

Collect global attributes from NetCDF, Zarr, or Dataset object.

`miranda.io.utils.get_time_attrs(file_or_dataset: str | os.PathLike | xr.Dataset)`

Determine attributes related to time dimensions.

`miranda.io.utils.name_output_file(ds_or_dict: Dataset | dict[str, str], output_format: str) → str`

Name an output file based on facets within a Dataset or a dictionary.

Parameters

- **ds_or_dict** (*xr.Dataset or dict*) – A miranda-converted Dataset or a dictionary containing the appropriate facets.
- **output_format** (*{“netcdf”, “zarr”}*) – Output filetype to be used for generating filename suffix.

Returns

str

Notes

If using a dictionary, the following keys must be set: * “variable”, “frequency”, “institution”, “time_start”, “time_end”.

`miranda.io.utils.sort_variables(files: list[pathlib.Path], variables: Sequence[str]) → dict[str, list[pathlib.Path]]`

Sort all variables within supplied files for treatment.

Parameters

- **files** (*list of Path*)
- **variables** (*sequence of str*)

Returns

dict[str, list[Path]]

miranda.ncar package

NCAR Download module

`miranda.ncar.cordex_aws_calendar_correction(ds) → Dataset | None`

AWS-stored CORDEX datasets are all on the same standard calendar, this converts the data back to the original calendar, removing added NaNs.

Credit: Pascal Bourgault (@aulemahal)

`miranda.ncar.cordex_aws_download(target_folder: str | Path, *, search: dict[str, str | list[str]], correct_times: bool = False, domain: str | None = None)`

Download CORDEX interpolated grid for North America from Amazon S3.

Submodules

miranda.ncar._aws_cordex module

`miranda.ncar._aws_cordex.cordex_aws_calendar_correction(ds) → Dataset | None`

AWS-stored CORDEX datasets are all on the same standard calendar, this converts the data back to the original calendar, removing added NaNs.

Credit: Pascal Bourgault (@aulemahal)

`miranda.ncar._aws_cordex.cordex_aws_download(target_folder: str | Path, *, search: dict[str, str | list[str]], correct_times: bool = False, domain: str | None = None)`

Download CORDEX interpolated grid for North America from Amazon S3.

miranda.remote package

Remote Operations module.

`class miranda.remote.Connection(username: str | Path | None = None, host: str | Path | None = None, protocol: str = 'sftp', *args, **kwargs)`

Bases: object

Connection contextualise class.

`connect(**kwargs)`

Connect to a remote server with credential prompts.

`update(**kwargs)`

Update connection keyword arguments.

Warning: Credentials are not encrypted.

`miranda.remote.archive_database(source: Path | str | list, common_path: Path | str, destination: Path | str, file_suffixes: str = '.nc', server: str | None = None, username: str | None = None, project_name: str | None = None, overwrite: bool = False, compression: bool = False, recursive: bool = False, use_grouping: bool = True, use_subdirectories: bool = True, dry_run: bool = False) → None`

Given a source, destination, and dependent on file size limit, create tarfile archives and transfer files to another server for backup purposes

```
miranda.remote.create_archive(source_files: list[str | os.PathLike], destination: str | os.PathLike, transport:
    SCPClient | SFTPClient | fabric.Connection | miranda.remote.Connection |
    None = None, delete: bool = True, compression: bool = False, recursive:
    bool = True) → None
```

Create an archive from source files and transfer to another location (remote or local).

Parameters

- **source_files** (*list of str or os.PathLike*)
- **destination** (*str or os.PathLike*)
- **transport** (*SCPClient or SFTPClient or fabric.Connection or miranda.remote.Connection, optional*)
- **delete** (*bool*)
- **compression** (*bool*)
- **recursive** (*bool*)

Returns

None

```
miranda.remote.create_remote_directory(directory: str | os.PathLike, transport: SSHClient |
    fabric.Connection | miranda.remote.Connection) → None
```

Call “mkdir -p” function to create a folder structure over SFTP/SSH and wait for confirmation before continuing.

Parameters

- **directory** (*str or os.PathLike*)
- **transport** (*SSHClient or fabric.Connection or miranda.remote.Connection*)

Returns

None

```
miranda.remote.delete_by_date(*, source: str | Path, year: int | None = None, month: int | None = None, day:
    int | None = None, pattern: str | None = None, server: str | Path | None =
    None, user: str | None = None, password: str | None = None, date_object:
    date | None = None) → None
```

Remove a selection of files based on a given date of last modification.

Parameters

- **source** (*str or Path*)
- **year** (*int, optional*)
- **month** (*int, optional*)
- **day** (*int, optional*)
- **pattern** (*str, optional*)
- **server** (*str or Path, optional*)
- **user** (*str, optional*)
- **password** (*str, optional*)
- **date_object** (*date, optional*)

Returns*None*

```
miranda.remote.delete_by_variable(*, target: str | Path | list[str | pathlib.Path] | generator | None = None,
                                variables: list[str], server: str | Path | None = None, user: str | None =
                                None, password: str | None = None, file_suffix: str | None = None,
                                delete: bool = False) → None
```

Delete according to variable name.

Given target location(s), a list of variables and a server address, perform a glob search and delete file names starting with the variables identified

Parameters

- **target** (*str, Path, list of str or Path, or GeneratorType*)
- **variables** (*list of str*)
- **server** (*str or Path, optional*)
- **user** (*str, optional*)
- **password** (*str, optional*)
- **file_suffix** (*str, optional*)
- **delete** (*bool*)

Returns*None*

```
miranda.remote.delete_duplicates(*, source: str | Path, target: str | Path, server: str | Path | None = None,
                                user: str | None = None, password: str | None = None, pattern: str | None
                                = None, delete_target_duplicates: bool = False) → None
```

Parameters

- **source** (*str or Path*)
- **target** (*str or Path*)
- **server** (*str or Path, optional*)
- **user** (*str*)
- **password** (*str*)
- **pattern** (*str*)
- **delete_target_duplicates** (*bool*)

Returns*None*

```
miranda.remote.file_emptyier(*, file_list: list[str | pathlib.Path] | generator) → None
```

Open and overwrite a list of file paths in order to delete data while preserving the file name.

Parameters

file_list (*list of str or Path, or GeneratorType*) – List of files to be overwritten

Returns*None*

```
miranda.remote.transfer_file(source_file: str | os.PathLike, destination_file: str | os.PathLike, transport:
                             SCPClient | SFTPClient | fabric.Connection | miranda.remote.Connection |
                             None = None) → bool
```

Transfer file from one location (remote or local) to another.

Parameters

- **source_file** (*str or os.PathLike*)
- **destination_file** (*str or os.PathLike*)
- **transport** (*SCPClient or SFTPClient or fabric.Connection or miranda.remote.Connection, optional*)

Returns

bool

Submodules

miranda.remote.archiver module

Archive Module.

```
miranda.remote.archiver.archive_database(source: Path | str | list, common_path: Path | str, destination:
                                          Path | str, file_suffixes: str = '.nc', server: str | None = None,
                                          username: str | None = None, project_name: str | None =
                                          None, overwrite: bool = False, compression: bool = False,
                                          recursive: bool = False, use_grouping: bool = True,
                                          use_subdirectories: bool = True, dry_run: bool = False) →
                                          None
```

Given a source, destination, and dependent on file size limit, create tarfile archives and transfer files to another server for backup purposes

miranda.remote.connect module

Remote Connection Operations module.

```
class miranda.remote.connect.Connection(username: str | Path | None = None, host: str | Path | None =
                                         None, protocol: str = 'sftp', *args, **kwargs)
```

Bases: object

Connection contextualise class.

connect(**kwargs)

Connect to a remote server with credential prompts.

update(**kwargs)

Update connection keyword arguments.

Warning: Credentials are not encrypted.

miranda.remote.ops module

Remote Operations module.

```
miranda.remote.ops.create_archive(source_files: list[str | os.PathLike], destination: str | os.PathLike,
                                  transport: SCPClient | SFTPClient | fabric.Connection |
                                  miranda.remote.Connection | None = None, delete: bool = True,
                                  compression: bool = False, recursive: bool = True) → None
```

Create an archive from source files and transfer to another location (remote or local).

Parameters

- **source_files** (*list of str or os.PathLike*)
- **destination** (*str or os.PathLike*)
- **transport** (*SCPClient or SFTPClient or fabric.Connection or miranda.remote.Connection, optional*)
- **delete** (*bool*)
- **compression** (*bool*)
- **recursive** (*bool*)

Returns

None

```
miranda.remote.ops.create_remote_directory(directory: str | os.PathLike, transport: SSHClient |
                                           fabric.Connection | miranda.remote.Connection) → None
```

Call “mkdir -p” function to create a folder structure over SFTP/SSH and wait for confirmation before continuing.

Parameters

- **directory** (*str or os.PathLike*)
- **transport** (*SSHClient or fabric.Connection or miranda.remote.Connection*)

Returns

None

```
miranda.remote.ops.transfer_file(source_file: str | os.PathLike, destination_file: str | os.PathLike,
                                  transport: SCPClient | SFTPClient | fabric.Connection |
                                  miranda.remote.Connection | None = None) → bool
```

Transfer file from one location (remote or local) to another.

Parameters

- **source_file** (*str or os.PathLike*)
- **destination_file** (*str or os.PathLike*)
- **transport** (*SCPClient or SFTPClient or fabric.Connection or miranda.remote.Connection, optional*)

Returns

bool

miranda.remote.remove module

Remote File Removal Operations module.

```
miranda.remote.remove.delete_by_date(*, source: str | Path, year: int | None = None, month: int | None =
                                     None, day: int | None = None, pattern: str | None = None, server:
                                     str | Path | None = None, user: str | None = None, password: str |
                                     None = None, date_object: date | None = None) → None
```

Remove a selection of files based on a given date of last modification.

Parameters

- **source** (*str or Path*)
- **year** (*int, optional*)
- **month** (*int, optional*)
- **day** (*int, optional*)
- **pattern** (*str, optional*)
- **server** (*str or Path, optional*)
- **user** (*str, optional*)
- **password** (*str, optional*)
- **date_object** (*date, optional*)

Returns

None

```
miranda.remote.remove.delete_by_variable(*, target: str | Path | list[str | pathlib.Path] | generator | None =
                                         None, variables: list[str], server: str | Path | None = None,
                                         user: str | None = None, password: str | None = None,
                                         file_suffix: str | None = None, delete: bool = False) → None
```

Delete according to variable name.

Given target location(s), a list of variables and a server address, perform a glob search and delete file names starting with the variables identified

Parameters

- **target** (*str, Path, list of str or Path, or GeneratorType*)
- **variables** (*list of str*)
- **server** (*str or Path, optional*)
- **user** (*str, optional*)
- **password** (*str, optional*)
- **file_suffix** (*str, optional*)
- **delete** (*bool*)

Returns

None

```
miranda.remote.remove.delete_duplicates(*, source: str | Path, target: str | Path, server: str | Path | None =
                                         None, user: str | None = None, password: str | None = None,
                                         pattern: str | None = None, delete_target_duplicates: bool =
                                         False) → None
```

Parameters

- **source** (*str or Path*)
- **target** (*str or Path*)
- **server** (*str or Path, optional*)
- **user** (*str*)
- **password** (*str*)
- **pattern** (*str*)
- **delete_target_duplicates** (*bool*)

Returns*None*

`miranda.remote.remove.file_emptier(*, file_list: list[str | pathlib.Path] | generator) → None`

Open and overwrite a list of file paths in order to delete data while preserving the file name.

Parameters

file_list (*list of str or Path, or GeneratorType*) – List of files to be overwritten

Returns*None***miranda.structure package**

Data Structure module.

`miranda.structure.build_path_from_schema(facets: dict, output_folder: str | PathLike, schema: str | PathLike | dict | None = None, top_folder: str = 'datasets', validate: bool = True) → Path | None`

Build a filepath based on a valid data schema.

Parameters

- **facets** (*dict*) – Facets for a given dataset.
- **output_folder** (*str or os.PathLike*) – Parent folder on which to extend the filetree structure.
- **schema** (*str or os.PathLike, optional*) – Path to YAML schematic of database structure. If None, will use Ouranos schema.
- **top_folder** (*str*) – Top-level of supplied schema, used for validation purposes. Default: “datasets”.
- **validate** (*bool*) – Run facets-validation checks over given file. Default: True.

Returns*Path or None*

`miranda.structure.create_version_hash_files(input_files: str | PathLike | list[str | os.PathLike] | generator | None = None, facet_dict: dict | None = None, verify_hash: bool = False) → None`

Create version hashes based on files or a facets dictionary.

Parameters

- **input_files** (*str, os.PathLike, list of str or os.PathLike, or GeneratorType*)
- **facet_dict** (*dict, optional*)

- **verify_hash** (*bool*)

Returns

None

`miranda.structure.structure_datasets(input_files: str | PathLike | list[str | os.PathLike] | generator, output_folder: str | PathLike, *, project: str | None = None, guess: bool = True, dry_run: bool = False, method: str = 'copy', make_dirs: bool = False, set_version_hashes: bool = False, verify_hashes: bool = False, suffix: str = 'nc') → dict[pathlib.Path, pathlib.Path]`

Parameters

- **input_files** (*str, Path, list of str or Path, or GeneratorType*) – Files to be sorted.
- **output_folder** (*str or Path*) – The desired location for the folder-tree.
- **project** (*{“cordex”, “cmip5”, “cmip6”, “isimip-ft”, “pcic-candcs-u6”, “converted”}*, *optional*) – Project used to parse the facets of all supplied datasets. If not supplied, will attempt parsing with all available data categories for each file (slow) unless *guess* is True.
- **guess** (*bool*) – If project not supplied, suggest to decoder that activity is the same for all input_files. Default: True.
- **dry_run** (*bool*) – Prints changes that would have been made without performing them. Default: False.
- **method** (*{“move”, “copy”}*) – Method to transfer files to intended location. Default: “move”.
- **make_dirs** (*bool*) – Make folder tree if it does not already exist. Default: False.
- **set_version_hashes** (*bool*) – Make an accompanying file with version in filename and sha256sum in contents. Default: False.
- **verify_hashes** (*bool*) – Ensure that any existing sha256sum files correspond with companion file. Raise on error. Default: False.
- **suffix** (*{“nc”, “zarr”}*) – If “zarr”, will perform a ‘glob’ with provided pattern. Otherwise, will perform an ‘rglob’ (recursive) operation.

Returns

dict[Path, Path]

Submodules

`miranda.structure._structure` module

`miranda.structure._structure.build_path_from_schema(facets: dict, output_folder: str | PathLike, schema: str | PathLike | dict | None = None, top_folder: str = 'datasets', validate: bool = True) → Path | None`

Build a filepath based on a valid data schema.

Parameters

- **facets** (*dict*) – Facets for a given dataset.
- **output_folder** (*str or os.PathLike*) – Parent folder on which to extend the filetree structure.

- **schema** (*str* or *os.PathLike*, optional) – Path to YAML schematic of database structure. If None, will use Ouranos schema.
- **top_folder** (*str*) – Top-level of supplied schema, used for validation purposes. Default: “datasets”.
- **validate** (*bool*) – Run facets-validation checks over given file. Default: True.

Returns*Path* or *None*

```
miranda.structure._structure.create_version_hash_files(input_files: str | PathLike | list[str |
os.PathLike] | generator | None = None,
facet_dict: dict | None = None, verify_hash:
bool = False) → None
```

Create version hashes based on files or a facets dictionary.

Parameters

- **input_files** (*str*, *os.PathLike*, list of *str* or *os.PathLike*, or *GeneratorType*)
- **facet_dict** (*dict*, optional)
- **verify_hash** (*bool*)

Returns*None*

```
miranda.structure._structure.structure_datasets(input_files: str | PathLike | list[str | os.PathLike] |
generator, output_folder: str | PathLike, *, project: str
| None = None, guess: bool = True, dry_run: bool =
False, method: str = 'copy', make_dirs: bool = False,
set_version_hashes: bool = False, verify_hashes:
bool = False, suffix: str = 'nc') → dict[pathlib.Path,
pathlib.Path]
```

Parameters

- **input_files** (*str*, *Path*, list of *str* or *Path*, or *GeneratorType*) – Files to be sorted.
- **output_folder** (*str* or *Path*) – The desired location for the folder-tree.
- **project** ({“cordex”, “cmip5”, “cmip6”, “isimip-ft”, “pcic-candcs-u6”, “converted”}, optional) – Project used to parse the facets of all supplied datasets. If not supplied, will attempt parsing with all available data categories for each file (slow) unless *guess* is True.
- **guess** (*bool*) – If project not supplied, suggest to decoder that activity is the same for all input_files. Default: True.
- **dry_run** (*bool*) – Prints changes that would have been made without performing them. Default: False.
- **method** ({“move”, “copy”}) – Method to transfer files to intended location. Default: “move”.
- **make_dirs** (*bool*) – Make folder tree if it does not already exist. Default: False.
- **set_version_hashes** (*bool*) – Make an accompanying file with version in filename and sha256sum in contents. Default: False.
- **verify_hashes** (*bool*) – Ensure that any existing sha256sum files correspond with companion file. Raise on error. Default: False.

- **suffix** (*{ "nc", "zarr" }*) – If “zarr”, will perform a ‘glob’ with provided pattern. Otherwise, will perform an ‘rglob’ (recursive) operation.

Returns

dict[Path, Path]

Submodules

miranda.cv module

Controlled Vocabulary module.

miranda.data module

Database Management module.

```
class miranda.data.DataBase(source, *, destination: Path | str | None = None, common_path: Path | str | None = None, file_pattern: str | list[str] = '*.nc', project_name: str | None = None, recursive: bool = True)
```

Bases: object

Database management class.

archive()

Not yet implemented.

```
group_by(*, common_path: Path | str | None = None, subdirectories: bool = True, dates: bool = True, size: int = 10737418240)
```

Grouping meta-function.

Notes

Not yet implemented.

items()

Show items.

keys()

Show keys.

```
target(target: Path | str)
```

Target directory or server address.

transfer()

Not yet implemented.

values()

Show values.

miranda.scripting module

Scripting Helpers module.

miranda.storage module

Disk space management

Classes:

- `DiskSpaceError` - the exception raised on failure.
- `FileMeta` - file and its size.
- `StorageState` - storage capacity and availability of a medium.

Functions:

- `total_size()` - get total size of a list of files.
- `size_division()` - divide files based on number and size restrictions.

exception `miranda.storage.DiskSpaceError`

Bases: `Exception`

`DiskSpaceError` Exception.

class `miranda.storage.FileMeta(path: str, size: int = -1)`

Bases: `object`

File path and size.

```
django = {'path': ['CharField', 'max_length=512'], 'size': ['IntegerField',
'null=True', 'blank=True']}
```

class `miranda.storage.StorageState(base_path, capacity=-1, used_space=-1, free_space=-1)`

Bases: `object`

Information regarding the storage capacity of a disk.

`miranda.storage.file_size(file_path_or_bytes_or_dict: Path | str | int | list[str | pathlib.Path] | generator | dict[str, pathlib.Path | list[pathlib.Path]]) → int`

Return size of object in bytes.

Parameters

file_path_or_bytes_or_dict (*Path or str or int, list of str or Path, GeneratorType, or dict[str, Path or list of Path]*)

Returns

int

`miranda.storage.report_file_size(file_path_or_bytes_or_dict: Path | str | int | list[str | pathlib.Path] | generator | dict[str, pathlib.Path | list[pathlib.Path]], use_binary: bool = True, significant_digits: int = 2) → str`

Report file size in a human-readable format.

This function will parse the contents of a list or generator of files and return the size in bytes of a file or a list of files in pretty formatted text.

Parameters

- **file_path_or_bytes_or_dict** (*Path or str or int, list of str or Path, GeneratorType, or dict[str, Path or list of Path]*)
- **use_binary** (*bool*)
- **significant_digits** (*int*)

`miranda.storage.size_division(files_to_divide: list | FileMeta | Path, size_limit: int = 0, file_limit: int = 0, check_name_repetition: bool = False, preserve_order: bool = False) → list[list]`

Divide files according to size and number limits.

Parameters

- **files_to_divide** (*list of str or Path, FileMeta, Path*) – Files to be sorted.
- **size_limit** (*int*) – Size limit of divisions in bytes. Default: 0 (no limit).
- **file_limit** (*int*) – Number of files limit of divisions. Default: 0 (no limit).
- **check_name_repetition** (*bool*) – Flag to prevent file name repetitions. Default: False.
- **preserve_order** (*bool*) – Flag to force files to be restored in the order they are given. Default: False.

Returns

list[list] – list of divisions (each division is a list of FileMeta objects).

`miranda.storage.size_evaluation(file_list: list[str | miranda.storage.FileMeta | pathlib.Path]) → int`

Total size of files.

Parameters

file_list (*list of str or Path or FileMeta*)

Returns

int – total size of files in bytes.

miranda.units module

Special Time Units-Handling submodule.

`miranda.units.get_time_frequency(d: Dataset, expected_period: str | None = None, minimum_continuous_period: str = '1M') → tuple[list[int | str], str]`

Try to understand the Dataset frequency.

If it can't be inferred with `xarray.infer_freq()` it tries to: - look for a “freq” attrs in the global or time variable attributes. - infer monthly frequency if all time steps are between 27 and 32 days

In the event that an *expected_period* is supplied, special handling will be called allowing for determining data that may be internally discontinuous (e.g. discontinuous overall, but continuous for *minimum_continuous_period*). This is provided for instances where input data in a multifile dataset is sparse.

Parameters

- **d** (*xr.Dataset*) – An xarray.Dataset.
- **expected_period** (*str*) – An xarray-compatible time period (e.g. “1H”, “1D”, “7D”, “1M”, “1A”). The time period expected of the input dataset. The “1M” period is specially-handled.
- **minimum_continuous_period** (*str*) – An xarray-compatible time period (e.g. “1H”, “1D”, “7D”, “1M”, “1A”). The minimum expected granular period that data should have continuous values for. The “1M” period is specially-handled.

Returns

- **offset** (*list of int or str*) – The offset a list of (multiplier, base)
- **offset_meaning** (*str*) – The offset meaning (single word)

`miranda.units.parse_offset(freq: str) → Sequence[str]`

Parse an offset string.

Parse a frequency offset and, if needed, convert to cftime-compatible components.

Parameters

freq (*str*) – Frequency offset.

Returns

- **multiplier** (*int*) – Multiplier of the base frequency. “[n]W” is always replaced with “[7n]D”, as xarray doesn’t support “W” for cftime indexes.
- **offset_base** (*str*) – Base frequency. “Y” is always replaced with “A”.
- **is_start_anchored** (*bool*) – Whether coordinates of this frequency should correspond to the beginning of the period (*True*) or its end (*False*). Can only be *False* when base is A, Q or M; in other words, xclim assumes frequencies finer than monthly are all start-anchored.
- **anchor** (*str or None*) – Anchor date for bases A or Q. As xarray doesn’t support “W”, neither does xclim (anchor information is lost when given).

miranda.utils module

Miscellaneous Helper Utilities module.

class `miranda.utils.HiddenPrints`

Bases: `object`

Special context manager for hiding print statements.

Notes

Solution from <https://stackoverflow.com/a/45669280/7322852> Credit to Alexander C (<https://stackoverflow.com/users/2039471/alexander-c>) CC-BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>)-

`miranda.utils.chunk_iterables(iterable: Sequence, chunk_size: int) → Iterable`

Generate lists of *chunk_size* elements from *iterable*.

Notes

Adapted from eidord (2012) <https://stackoverflow.com/a/12797249/7322852> (<https://creativecommons.org/licenses/by-sa/4.0/>)

`miranda.utils.generic_extract_archive(resources: str | Path | list[bytes | str | pathlib.Path], output_dir: Path | str | None = None) → list[pathlib.Path]`

Extract archives (tar/zip) to a working directory.

Parameters

- **resources** (*str or Path or list of bytes or str or Path*) – list of archive files (if netCDF files are in list, they are passed and returned as well in the return).

- **output_dir** (*str or Path, optional*) – string or Path to a working location (default: temporary folder).

Returns

list – List of original or of extracted files

`miranda.utils.list_paths_with_elements(base_paths: str | list[str], elements: list[str]) → list[dict]`

List a given path structure.

Parameters

- **base_paths** (*list of str*) – List of paths from which to start the search.
- **elements** (*list of str*) – Ordered list of the expected elements.

Returns

list of dict – The keys are ‘path’ and each of the members of the given elements, the path is the absolute path.

Notes

Suppose you have the following structure: `/base_path/{color}/{shape}` The resulting list would look like:

```
[{'path': '/base_path/red/square', 'color': 'red', 'shape': 'square'},
 {'path': '/base_path/red/circle', 'color': 'red', 'shape': 'circle'},
 {'path': '/base_path/blue/triangle', 'color': 'blue', 'shape': 'triangle'},
 ...]
```

Obviously, ‘path’ should not be in the input list of elements.

`miranda.utils.publish_release_notes(style: str = 'md', file: PathLike | StringIO | TextIO | None = None) → str | None`

Format release history in Markdown or ReStructuredText.

Parameters

- **style** (*{“rst”, “md”}*) – Use ReStructuredText formatting or Markdown. Default: Markdown.
- **file** (*{os.PathLike, StringIO, TextIO}, optional*) – If provided, prints to the given file-like object. Otherwise, returns a string.

Returns

str, optional

Notes

This function is solely for development purposes.

`miranda.utils.single_item_list(iterable: Iterable) → bool`

Ascertain whether a list has exactly one entry.

See: <https://stackoverflow.com/a/16801605/7322852>

Parameters

iterable (*Iterable*)

Returns

bool

`miranda.utils.working_directory(directory: str | Path) → None`

Change the working directory within a context object.

This function momentarily changes the working directory within the context and reverts to the file working directory when the code block it is acting upon exits

Parameters

directory (*str or pathlib.Path*)

Returns

None

miranda.validators module

Data Validation module.

`miranda.validators.url_validate(target: str) → Match[str] | None`

Validate whether a supplied URL is reliably written.

Parameters

target (*str*)

References

<https://stackoverflow.com/a/7160778/7322852>

3.2 Feedback

If you have any suggestions or questions about **Miranda** feel free to email me at smith.trevorj@ouranos.ca.

If you encounter any errors or problems with **Miranda**, please let me know! Open an Issue at the GitHub <https://github.com/Ouranosinc/miranda> main repository.

PYTHON MODULE INDEX

m

- miranda, 17
- miranda.archive, 18
- miranda.archive._groupings, 19
- miranda.archive._selection, 20
- miranda.convert, 21
- miranda.convert._aggregation, 25
- miranda.convert._data_corrections, 25
- miranda.convert._data_definitions, 27
- miranda.convert._reconstruction, 29
- miranda.convert.deh, 30
- miranda.convert.eccc, 30
- miranda.convert.eccc_rdrs, 30
- miranda.convert.ecmwf, 31
- miranda.convert.hq, 31
- miranda.convert.melcc, 32
- miranda.convert.utils, 34
- miranda.cv, 68
- miranda.data, 68
- miranda.decode, 35
- miranda.decode._decoder, 36
- miranda.decode._time, 37
- miranda.eccc, 37
- miranda.eccc._homogenized, 39
- miranda.eccc._raw, 40
- miranda.eccc._summaries, 41
- miranda.eccc._support_rvt, 42
- miranda.eccc._utils, 45
- miranda.ecmwf, 45
- miranda.ecmwf._era5, 47
- miranda.ecmwf._tigge, 48
- miranda.gis, 48
- miranda.gis._domains, 49
- miranda.io, 50
- miranda.io._input, 53
- miranda.io._output, 54
- miranda.io._rechunk, 56
- miranda.io.utils, 57
- miranda.ncar, 59
- miranda.ncar._aws_cortex, 59
- miranda.remote, 59
- miranda.remote.archiver, 62
- miranda.remote.connect, 62
- miranda.remote.ops, 63
- miranda.remote.remove, 64
- miranda.scripting, 69
- miranda.storage, 69
- miranda.structure, 65
- miranda.structure._structure, 66
- miranda.units, 70
- miranda.utils, 71
- miranda.validators, 73

A

`absolute()` (*miranda.eccc._support_rvt.Path* method), 42

`add_ar6_regions()` (*in module miranda.gis*), 48

`add_ar6_regions()` (*in module miranda.gis._domains*), 49

`aggregate()` (*in module miranda.convert*), 21

`aggregate()` (*in module miranda.convert._aggregation*), 25

`aggregate_stations()` (*in module miranda.eccc*), 37

`aggregate_stations()` (*in module miranda.eccc._raw*), 40

`aggregations_possible()` (*in module miranda.convert*), 21

`aggregations_possible()` (*in module miranda.convert._aggregation*), 25

`archive()` (*miranda.data.DataBase* method), 68

`archive()` (*miranda.DataBase* method), 17

`archive_database()` (*in module miranda.remote*), 59

`archive_database()` (*in module miranda.remote.archiver*), 62

B

`build_path_from_schema()` (*in module miranda.structure*), 65

`build_path_from_schema()` (*in module miranda.structure._structure*), 66

C

`cf_ahccd_metadata()` (*in module miranda.eccc._utils*), 45

`cf_station_metadata()` (*in module miranda.eccc._utils*), 45

`chmod()` (*miranda.eccc._support_rvt.Path* method), 42

`chunk_iterables()` (*in module miranda.utils*), 71

`concat()` (*in module miranda.convert.melcc*), 32

`concat_rechunk_zarr()` (*in module miranda.io*), 50

`concat_rechunk_zarr()` (*in module miranda.io._output*), 54

`connect()` (*miranda.remote.connect.Connection* method), 62

`connect()` (*miranda.remote.Connection* method), 59

`Connection` (*class in miranda.remote*), 59

`Connection` (*class in miranda.remote.connect*), 62

`convert_ahccd()` (*in module miranda.eccc*), 37

`convert_ahccd()` (*in module miranda.eccc._homogenized*), 39

`convert_ahccd_fwf_files()` (*in module miranda.eccc*), 38

`convert_ahccd_fwf_files()` (*in module miranda.eccc._homogenized*), 40

`convert_canswe()` (*in module miranda.convert.eccc*), 30

`convert_flat_files()` (*in module miranda.eccc*), 38

`convert_flat_files()` (*in module miranda.eccc._raw*), 40

`convert_mdb()` (*in module miranda.convert.melcc*), 33

`convert_melcc_obs()` (*in module miranda.convert.melcc*), 33

`convert_rdrs()` (*in module miranda.convert.eccc_rdrs*), 30

`convert_snow_table()` (*in module miranda.convert.melcc*), 33

`cordex_aws_calendar_correction()` (*in module miranda.ncar*), 59

`cordex_aws_calendar_correction()` (*in module miranda.ncar._aws_cordex*), 59

`cordex_aws_download()` (*in module miranda.ncar*), 59

`cordex_aws_download()` (*in module miranda.ncar._aws_cordex*), 59

`create_archive()` (*in module miranda.remote*), 60

`create_archive()` (*in module miranda.remote.ops*), 63

`create_remote_directory()` (*in module miranda.remote*), 60

`create_remote_directory()` (*in module miranda.remote.ops*), 63

`create_version_hash_files()` (*in module miranda.structure*), 65

`create_version_hash_files()` (*in module miranda.structure._structure*), 67

`creation_date()` (*in module miranda.io.utils*), 57

`cwd()` (*miranda.eccc._support_rvt.Path* class method), 42

D

`daily_summaries_to_netcdf()` (in module `miranda.eccc`), 38

`daily_summaries_to_netcdf()` (in module `miranda.eccc.summaries`), 41

`DataBase` (class in `miranda`), 17

`DataBase` (class in `miranda.data`), 68

`dataset_conversion()` (in module `miranda.convert`), 21

`dataset_conversion()` (in module `miranda.convert.data_corrections`), 25

`dataset_corrections()` (in module `miranda.convert`), 22

`dataset_corrections()` (in module `miranda.convert.data_corrections`), 26

`date_parser()` (in module `miranda.convert.utils`), 34

`decode()` (`miranda.decode._decoder.Decoder` method), 36

`decode()` (`miranda.decode.Decoder` method), 35

`decode_ahccd_obs()` (`miranda.decode._decoder.Decoder` static method), 36

`decode_ahccd_obs()` (`miranda.decode.Decoder` static method), 35

`decode_cmip5()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_cmip5()` (`miranda.decode.Decoder` class method), 35

`decode_cmip6()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_cmip6()` (`miranda.decode.Decoder` class method), 35

`decode_converted()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_converted()` (`miranda.decode.Decoder` class method), 35

`decode_cordex()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_cordex()` (`miranda.decode.Decoder` class method), 35

`decode_eccc_obs()` (`miranda.decode._decoder.Decoder` static method), 36

`decode_eccc_obs()` (`miranda.decode.Decoder` static method), 35

`decode_espo_g6_e5l()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_espo_g6_e5l()` (`miranda.decode.Decoder` class method), 35

`decode_espo_g6_r2()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_espo_g6_r2()` (`miranda.decode.Decoder` class method), 35

`decode_isimip_ft()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_isimip_ft()` (`miranda.decode.Decoder` class method), 35

`decode_melcc_obs()` (`miranda.decode._decoder.Decoder` static method), 36

`decode_melcc_obs()` (`miranda.decode.Decoder` static method), 35

`decode_nex_gddp_cmip6()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_nex_gddp_cmip6()` (`miranda.decode.Decoder` class method), 35

`decode_pcic_candcs_u6()` (`miranda.decode._decoder.Decoder` class method), 36

`decode_pcic_candcs_u6()` (`miranda.decode.Decoder` class method), 35

`Decoder` (class in `miranda.decode`), 35

`Decoder` (class in `miranda.decode._decoder`), 36

`DecoderError`, 36, 37

`delayed_write()` (in module `miranda.io.utils`), 57

`delete_by_date()` (in module `miranda.remote`), 60

`delete_by_date()` (in module `miranda.remote.remove`), 64

`delete_by_variable()` (in module `miranda.remote`), 61

`delete_by_variable()` (in module `miranda.remote.remove`), 64

`delete_duplicates()` (in module `miranda.remote`), 61

`delete_duplicates()` (in module `miranda.remote.remove`), 64

`dims_conversion()` (in module `miranda.convert`), 22

`dims_conversion()` (in module `miranda.convert.data_corrections`), 26

`discover_data()` (in module `miranda.io`), 50

`discover_data()` (in module `miranda.io._input`), 53

`DiskSpaceError`, 69

`django` (`miranda.FileMeta` attribute), 18

`django` (`miranda.storage.FileMeta` attribute), 69

E

`exists()` (`miranda.eccc._support_rvt.Path` method), 42

`expanduser()` (`miranda.eccc._support_rvt.Path` method), 42

`extract_daily_summaries()` (in module `miranda.eccc`), 39

`extract_daily_summaries()` (in module `miranda.eccc.summaries`), 41

F

facets_table() (*miranda.decode._decoder.Decoder method*), 36
 facets_table() (*miranda.decode.Decoder method*), 35
 fetch_chunk_config() (*in module miranda.io*), 50
 fetch_chunk_config() (*in module miranda.io._rechunk*), 56
 file_emptier() (*in module miranda.remote*), 61
 file_emptier() (*in module miranda.remote.remove*), 65
 file_facets() (*miranda.decode._decoder.Decoder method*), 37
 file_facets() (*miranda.decode.Decoder method*), 35
 file_size() (*in module miranda.storage*), 69
 FileMeta (*class in miranda*), 18
 FileMeta (*class in miranda.storage*), 69
 find_filepaths() (*in module miranda.io*), 51
 find_filepaths() (*in module miranda.io._input*), 54
 find_version_hash() (*in module miranda.convert.utils*), 35

G

gather_agcfsr() (*in module miranda.convert*), 22
 gather_agcfsr() (*in module miranda.convert._data_definitions*), 27
 gather_agmerra() (*in module miranda.convert*), 22
 gather_agmerra() (*in module miranda.convert._data_definitions*), 27
 gather_eccc_stations() (*in module miranda.eccc._support_rvt*), 44
 gather_ecmwf() (*in module miranda.convert*), 22
 gather_ecmwf() (*in module miranda.convert._data_definitions*), 27
 gather_emdna() (*in module miranda.convert*), 22
 gather_emdna() (*in module miranda.convert._data_definitions*), 28
 gather_grnch() (*in module miranda.convert*), 23
 gather_grnch() (*in module miranda.convert._data_definitions*), 28
 gather_nex() (*in module miranda.convert*), 23
 gather_nex() (*in module miranda.convert._data_definitions*), 28
 gather_nrcan_gridded_obs() (*in module miranda.convert*), 23
 gather_nrcan_gridded_obs() (*in module miranda.convert._data_definitions*), 28
 gather_raw_rdrs_by_years() (*in module miranda.convert*), 23
 gather_raw_rdrs_by_years() (*in module miranda.convert._data_definitions*), 28
 gather_rdrs() (*in module miranda.convert*), 23
 gather_rdrs() (*in module miranda.convert._data_definitions*), 28
 gather_sc_earth() (*in module miranda.convert*), 24

gather_sc_earth() (*in module miranda.convert._data_definitions*), 29
 gather_wfdei_gem_capa() (*in module miranda.convert*), 24
 gather_wfdei_gem_capa() (*in module miranda.convert._data_definitions*), 29
 generic_extract_archive() (*in module miranda.utils*), 71
 get_chunks_on_disk() (*in module miranda.io.utils*), 58
 get_global_attrs() (*in module miranda.io.utils*), 58
 get_time_attrs() (*in module miranda.io.utils*), 58
 get_time_frequency() (*in module miranda.units*), 70
 glob() (*miranda.eccc._support_rvt.Path method*), 42
 group() (*miranda.eccc._support_rvt.Path method*), 42
 group_by() (*miranda.data.DataBase method*), 68
 group_by() (*miranda.DataBase method*), 17
 group_by_deciphered_date() (*in module miranda.archive*), 18
 group_by_deciphered_date() (*in module miranda.archive._groupings*), 19
 group_by_length() (*in module miranda.archive*), 18
 group_by_length() (*in module miranda.archive._groupings*), 20
 group_by_size() (*in module miranda.archive*), 19
 group_by_size() (*in module miranda.archive._groupings*), 20
 group_by_subdirectories() (*in module miranda.archive*), 19
 group_by_subdirectories() (*in module miranda.archive._groupings*), 20
 guess (*miranda.decode._decoder.Decoder attribute*), 37
 guess (*miranda.decode.Decoder attribute*), 35
 guess_project() (*in module miranda.decode*), 36
 guess_project() (*in module miranda.decode._decoder*), 37

H

hardlink_to() (*miranda.eccc._support_rvt.Path method*), 42
 HiddenPrints (*class in miranda.utils*), 71
 home() (*miranda.eccc._support_rvt.Path class method*), 42

I

is_block_device() (*miranda.eccc._support_rvt.Path method*), 42
 is_char_device() (*miranda.eccc._support_rvt.Path method*), 42
 is_dir() (*miranda.eccc._support_rvt.Path method*), 42
 is_fifo() (*miranda.eccc._support_rvt.Path method*), 43
 is_file() (*miranda.eccc._support_rvt.Path method*), 43

[is_mount\(\)](#) (*miranda.eccc._support_rvt.Path method*),
[43](#)
[is_socket\(\)](#) (*miranda.eccc._support_rvt.Path method*),
[43](#)
[is_symlink\(\)](#) (*miranda.eccc._support_rvt.Path method*), [43](#)
[items\(\)](#) (*miranda.data.DataBase method*), [68](#)
[items\(\)](#) (*miranda.DataBase method*), [18](#)
[iterdir\(\)](#) (*miranda.eccc._support_rvt.Path method*),
[43](#)

K

[keys\(\)](#) (*miranda.data.DataBase method*), [68](#)
[keys\(\)](#) (*miranda.DataBase method*), [18](#)

L

[lchmod\(\)](#) (*miranda.eccc._support_rvt.Path method*), [43](#)
[link_to\(\)](#) (*miranda.eccc._support_rvt.Path method*),
[43](#)
[list_paths_with_elements\(\)](#) (*in module mi-
 randa.utils*), [72](#)
[list_tables\(\)](#) (*in module miranda.convert.melcc*), [33](#)
[load_json_data_mappings\(\)](#) (*in module mi-
 randa.convert*), [24](#)
[load_json_data_mappings\(\)](#) (*in module mi-
 randa.convert._data_corrections*), [26](#)
[lstat\(\)](#) (*miranda.eccc._support_rvt.Path method*), [43](#)

M

[merge_converted_variables\(\)](#) (*in module mi-
 randa.eccc*), [39](#)
[merge_converted_variables\(\)](#) (*in module mi-
 randa.eccc._raw*), [41](#)
[merge_rechunk_zarrs\(\)](#) (*in module miranda.io*), [51](#)
[merge_rechunk_zarrs\(\)](#) (*in module mi-
 randa.io._output*), [54](#)
[metadata_conversion\(\)](#) (*in module miranda.convert*),
[24](#)
[metadata_conversion\(\)](#) (*in module mi-
 randa.convert._data_corrections*), [26](#)
[miranda](#)
 module, [17](#)
[miranda.archive](#)
 module, [18](#)
[miranda.archive._groupings](#)
 module, [19](#)
[miranda.archive._selection](#)
 module, [20](#)
[miranda.convert](#)
 module, [21](#)
[miranda.convert._aggregation](#)
 module, [25](#)
[miranda.convert._data_corrections](#)
 module, [25](#)
[miranda.convert._data_definitions](#)
 module, [27](#)
[miranda.convert._reconstruction](#)
 module, [29](#)
[miranda.convert.deh](#)
 module, [30](#)
[miranda.convert.eccc](#)
 module, [30](#)
[miranda.convert.eccc_rdrs](#)
 module, [30](#)
[miranda.convert.ecmwf](#)
 module, [31](#)
[miranda.convert.hq](#)
 module, [31](#)
[miranda.convert.melcc](#)
 module, [32](#)
[miranda.convert.utils](#)
 module, [34](#)
[miranda.cv](#)
 module, [68](#)
[miranda.data](#)
 module, [68](#)
[miranda.decode](#)
 module, [35](#)
[miranda.decode._decoder](#)
 module, [36](#)
[miranda.decode._time](#)
 module, [37](#)
[miranda.eccc](#)
 module, [37](#)
[miranda.eccc._homogenized](#)
 module, [39](#)
[miranda.eccc._raw](#)
 module, [40](#)
[miranda.eccc._summaries](#)
 module, [41](#)
[miranda.eccc._support_rvt](#)
 module, [42](#)
[miranda.eccc._utils](#)
 module, [45](#)
[miranda.ecmwf](#)
 module, [45](#)
[miranda.ecmwf._era5](#)
 module, [47](#)
[miranda.ecmwf._tigge](#)
 module, [48](#)
[miranda.gis](#)
 module, [48](#)
[miranda.gis._domains](#)
 module, [49](#)
[miranda.io](#)
 module, [50](#)
[miranda.io._input](#)
 module, [53](#)

miranda.io._output
 module, 54
 miranda.io._rechunk
 module, 56
 miranda.io.utils
 module, 57
 miranda.ncar
 module, 59
 miranda.ncar._aws_cortex
 module, 59
 miranda.remote
 module, 59
 miranda.remote.archiver
 module, 62
 miranda.remote.connect
 module, 62
 miranda.remote.ops
 module, 63
 miranda.remote.remove
 module, 64
 miranda.scripting
 module, 69
 miranda.storage
 module, 69
 miranda.structure
 module, 65
 miranda.structure._structure
 module, 66
 miranda.units
 module, 70
 miranda.utils
 module, 71
 miranda.validators
 module, 73
 mkdir() (*miranda.eccc._support_rvt.Path* method), 43
 module
 miranda, 17
 miranda.archive, 18
 miranda.archive._groupings, 19
 miranda.archive._selection, 20
 miranda.convert, 21
 miranda.convert._aggregation, 25
 miranda.convert._data_corrections, 25
 miranda.convert._data_definitions, 27
 miranda.convert._reconstruction, 29
 miranda.convert.deh, 30
 miranda.convert.eccc, 30
 miranda.convert.eccc_rdrs, 30
 miranda.convert.ecmwf, 31
 miranda.convert.hq, 31
 miranda.convert.melcc, 32
 miranda.convert.utils, 34
 miranda.cv, 68
 miranda.data, 68

miranda.decode, 35
 miranda.decode._decoder, 36
 miranda.decode._time, 37
 miranda.eccc, 37
 miranda.eccc._homogenized, 39
 miranda.eccc._raw, 40
 miranda.eccc._summaries, 41
 miranda.eccc._support_rvt, 42
 miranda.eccc._utils, 45
 miranda.ecmwf, 45
 miranda.ecmwf._era5, 47
 miranda.ecmwf._tigge, 48
 miranda.gis, 48
 miranda.gis._domains, 49
 miranda.io, 50
 miranda.io._input, 53
 miranda.io._output, 54
 miranda.io._rechunk, 56
 miranda.io.utils, 57
 miranda.ncar, 59
 miranda.ncar._aws_cortex, 59
 miranda.remote, 59
 miranda.remote.archiver, 62
 miranda.remote.connect, 62
 miranda.remote.ops, 63
 miranda.remote.remove, 64
 miranda.scripting, 69
 miranda.storage, 69
 miranda.structure, 65
 miranda.structure._structure, 66
 miranda.units, 70
 miranda.utils, 71
 miranda.validators, 73

N

name_output_file() (*in module miranda.io.utils*), 58

O

open() (*miranda.eccc._support_rvt.Path* method), 43
 open_csv() (*in module miranda.convert.hq*), 31
 open_txt() (*in module miranda.convert.deh*), 30
 owner() (*miranda.eccc._support_rvt.Path* method), 43

P

parse_offset() (*in module miranda.units*), 71
 parse_var_code() (*in module miranda.convert.melcc*), 33
 Path (*class in miranda.eccc._support_rvt*), 42
 prepare_chunks_for_ds() (*in module miranda.io*), 52
 prepare_chunks_for_ds() (*in module miranda.io._rechunk*), 56
 project (*miranda.decode._decoder.Decoder* attribute), 37
 project (*miranda.decode.Decoder* attribute), 36

`publish_release_notes()` (in module `miranda.utils`),
72

R

`rdrs_to_daily()` (in module `miranda.convert.eccc_rdrs`), 30

`read_bytes()` (`miranda.eccc._support_rvt.Path` method), 43

`read_definitions()` (in module `miranda.convert.melcc`), 33

`read_stations()` (in module `miranda.convert.melcc`),
34

`read_table()` (in module `miranda.convert.melcc`), 34

`read_text()` (`miranda.eccc._support_rvt.Path` method),
43

`readlink()` (`miranda.eccc._support_rvt.Path` method),
43

`reanalysis_processing()` (in module `miranda.convert._reconstruction`), 29

`rechunk_files()` (in module `miranda.io`), 52

`rechunk_files()` (in module `miranda.io._rechunk`), 56

`rename()` (`miranda.eccc._support_rvt.Path` method), 43

`rename_era5_files()` (in module `miranda.ecmwf`), 45

`rename_era5_files()` (in module `miranda.ecmwf._era5`), 47

`replace()` (`miranda.eccc._support_rvt.Path` method),
44

`report_file_size()` (in module `miranda.storage`), 69

`request_era5()` (in module `miranda.ecmwf`), 45

`request_era5()` (in module `miranda.ecmwf._era5`), 47

`request_tigge()` (in module `miranda.ecmwf`), 46

`request_tigge()` (in module `miranda.ecmwf._tigge`),
48

`resolve()` (`miranda.eccc._support_rvt.Path` method),
44

`rglob()` (`miranda.eccc._support_rvt.Path` method), 44

`rmdir()` (`miranda.eccc._support_rvt.Path` method), 44

S

`samefile()` (`miranda.eccc._support_rvt.Path` method),
44

`select_by_date_modified()` (in module `miranda.archive`), 19

`select_by_date_modified()` (in module `miranda.archive._selection`), 20

`single_item_list()` (in module `miranda.utils`), 72

`size_division()` (in module `miranda.storage`), 70

`size_evaluation()` (in module `miranda.storage`), 70

`sort_variables()` (in module `miranda.io.utils`), 58

`stat()` (`miranda.eccc._support_rvt.Path` method), 44

`StorageState` (class in `miranda`), 18

`StorageState` (class in `miranda.storage`), 69

`structure_datasets()` (in module `miranda.structure`),
66

`structure_datasets()` (in module `miranda.structure._structure`), 67

`subset_domain()` (in module `miranda.gis`), 48

`subset_domain()` (in module `miranda.gis._domains`),
49

`subsetting_domains()` (in module `miranda.gis`), 49

`subsetting_domains()` (in module `miranda.gis._domains`), 49

`symlink_to()` (`miranda.eccc._support_rvt.Path` method), 44

T

`target()` (`miranda.data.DataBase` method), 68

`target()` (`miranda.DataBase` method), 18

`threshold_mask()` (in module `miranda.convert`), 24

`threshold_mask()` (in module `miranda.convert._data_corrections`), 26

`tigge_convert()` (in module `miranda.convert.ecmwf`),
31

`touch()` (`miranda.eccc._support_rvt.Path` method), 44

`transfer()` (`miranda.data.DataBase` method), 68

`transfer()` (`miranda.DataBase` method), 18

`transfer_file()` (in module `miranda.remote`), 61

`transfer_file()` (in module `miranda.remote.ops`), 63

`translate_time_chunk()` (in module `miranda.io`), 52

`translate_time_chunk()` (in module `miranda.io._rechunk`), 57

U

`unlink()` (`miranda.eccc._support_rvt.Path` method), 44

`update()` (`miranda.remote.connect.Connection` method),
62

`update()` (`miranda.remote.Connection` method), 59

`url_validate()` (in module `miranda.validators`), 73

V

`values()` (`miranda.data.DataBase` method), 68

`values()` (`miranda.DataBase` method), 18

`variable_conversion()` (in module `miranda.convert`),
24

`variable_conversion()` (in module `miranda.convert._data_corrections`), 27

W

`working_directory()` (in module `miranda.utils`), 72

`write_bytes()` (`miranda.eccc._support_rvt.Path` method), 44

`write_dataset()` (in module `miranda.io`), 53

`write_dataset()` (in module `miranda.io._output`), 55

`write_dataset_dict()` (in module `miranda.io`), 53

`write_dataset_dict()` (in module `miranda.io._output`), 55

`write_text()` (`miranda.eccc._support_rvt.Path` method), 44